Configuring a Denodo Cluster with HAProxy Load Balancer

Revision 20200818

Denodo North America & APAC: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA
Denodo Iberia & Latino América: Montalbán 5, 28014 Madrid, Spain
Denodo EMEA: 21st Floor, Portland House, Bressenden Place, London SW1E 5RS. UK
Denodo DACH: Karlstraße 10, 80333 München. Germany
www.denodo.com

CONTENTS

1 INTRODUCTION

This document aims at explaining how to set up an active-active Denodo Cluster with 2 nodes behind the HAProxy load balancer. While the configuration is specific to the chosen load balancer, the concepts should be general enough to serve as guidelines for other load balancers configuration.

In detail, we are going to cover the following topics:

- Configuration of the load balancer
- Implementation of HTTP session persistence (or stickiness)
- Setup of the deployment scripts in the Solution Manager
- Considerations on the Denodo internal metadata catalogs in clustered deployments
- Testing recommendations

2 SOFTWARE VERSIONS

2.1 DENODO 7

Denodo Platform	7.0 20200205
HAProxy	HA-Proxy version 1.8.8-1ubuntu0.9 2019/12/02
Operating System	Ubuntu Server 18.04 on all the machines

2.2 DENODO 8

Denodo Platform	8.0
HAProxy	HA-Proxy version 2.0.13-2 2020/04/01 - https://haproxy.org/ (Denodo 8)
Operating System	Ubuntu Server 20.04 on all the Machines

3

4 CLUSTER ARCHITECTURE AND INTRODUCTION TO HAPROXY

The architecture discussed in this article is shown in the following diagram:

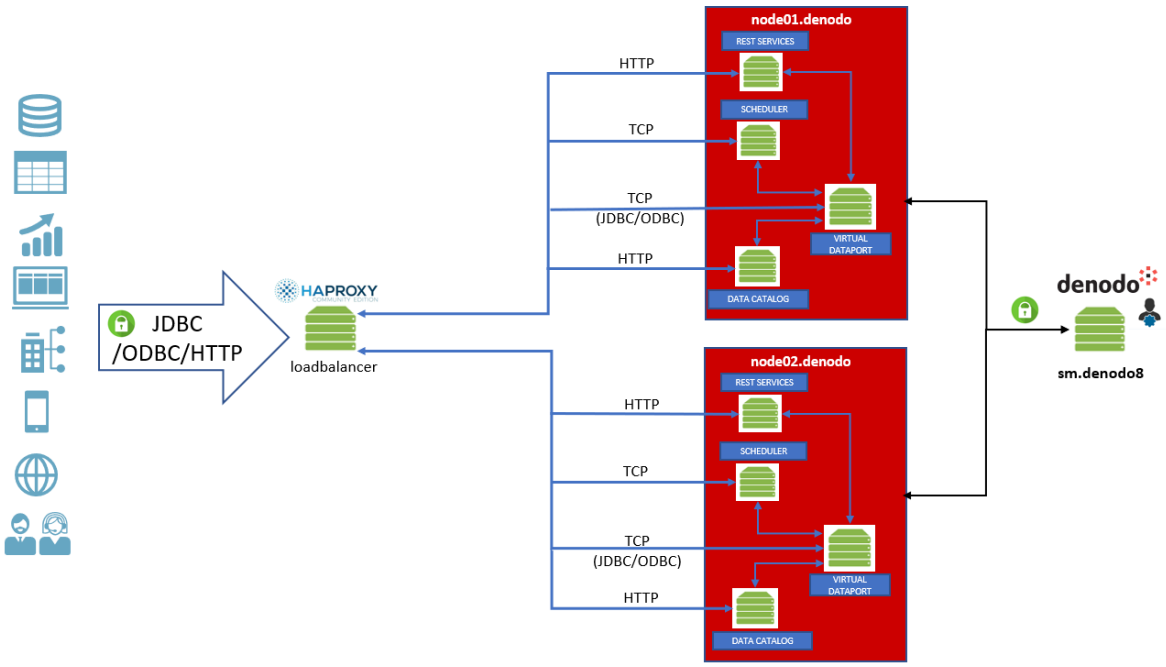


Figure 1: General Architecture with 2 Denodo Platform active nodes and a load balancer

Client applications (on the left) access the Denodo platform nodes (`node01.denodo8` and `node02.denodo8`) through the load balancer machine (`loadbalancer`). The two Denodo nodes are referenced in the Solution Manager (`sm.denodo8` on the right) with which they have normal interactions (license check, monitoring, promotion, ...). The bidirectional Solution Manager -> nodes traffic bypasses the HAProxy Load Balancer.

External (from client applications) and internal (between Denodo servers) communication is secured via SSL/TLS encryption. For Denodo 8, this is not just a typical recommendation for production deployments, but a requirement to be able to perform promotions in the Solution Manager. Indeed, starting from this version, to enable promotions the Denodo Security Token must be enabled in the involved servers and the Denodo Security Token in turn requires SSL/TLS.

In the following table, we can see all the components of the Denodo Platform that can be load balanced:

Component	Protocol	Default Port	Comments

Virtual DataPort - JDBC	TCP	9999	
Virtual DataPort - JDBC (RMI Factory Port)	TCP	9997	Denodo 7 only
Virtual DataPort - ODBC	TCP	9996	
Scheduler	TCP	8000	
Scheduler Index	TCP	9000	Not treated in this article
Data Catalog	HTTP	9090 (SSL: 9443)	
Scheduler Web Admin Tool	HTTP	9090 (SSL: 9443)	
RESTful Web Service	HTTP	9090 (SSL: 9443)	
REST Web Services	HTTP	9090 (SSL: 9443)	
Diagnostic & Monitoring Tool	HTTP	9090 (SSL: 9443)	

Components of the Denodo Platform that can be load balanced

This article does not address load balancing for the ITPilot modules (ITPilot Browser Pool, ITPilot Verification Server, ITPilot PDF Conversion Server) and the Scheduler Index Server.

The load balancer is based on HAProxy, a free and open source software that provides a high availability load balancer and proxy server for TCP and HTTP-based applications that spreads requests across multiple servers. It is written in C and has a reputation for being fast and efficient in terms of processor and memory usage.

5 SETUP OF HAPROXY IN THE LOAD BALANCER MACHINE

Install the package via `apt` (you may want to install it from source for the latest version):

```
$ sudo apt install haproxy
```

The main configuration file is `/etc/haproxy/haproxy.cfg` and the installation creates a `systemd` service:

```
$ systemctl status haproxy
● haproxy.service - HAProxy Load Balancer
      Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor
preset: enabled)
      Active: active (running) since Mon 2020-08-10 12:36:00 CEST; 5h 55min
ago
      Docs: man:haproxy(1)
            file:/usr/share/doc/haproxy/configuration.txt.gz
      Process: 48519 ExecStartPre=/usr/sbin/haproxy -f $CONFIG -c -q
$EXTRAOPTS (code=exited, status=0/SUCCESS)
   Main PID: 48530 (haproxy)
      Tasks: 2 (limit: 1075)
      Memory: 3.9M
      CGroup: /system.slice/haproxy.service
            ├─48530 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p
/run/haproxy.pid -S /run/haproxy-master.sock
            └─48531 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p
/run/haproxy.pid -S /run/haproxy-master.sock
```

Every time you modify the configuration you have to restart the service in order to get it taken into account.

```
$ sudo systemctl restart haproxy
```

Before restarting you may want to check that the configuration file is valid

```
$ haproxy -f /etc/haproxy/haproxy.cfg -c
Configuration file is valid
```

5.1

5.2 HAPROXY CONFIGURATION FILE

Let's have a look at the configuration file

```
global
      log 127.0.0.1 local2 info
      chroot /var/lib/haproxy
      pidfile /var/run/haproxy.pid
      maxconn 256
      user haproxy
      group haproxy
      daemon
      ## enabling the HAProxy Runtime API
      stats socket ipv4@127.0.0.1:7777 level admin
      stats socket /var/run/haproxy.sock mode 666 level admin
      stats timeout 2m

defaults
      mode        http
      log         global
      option            httplog
      timeout     connect 10s
      timeout     client 30s
      timeout     server 30s

## http/9090 => Proxy for Apache Tomcat Based Web Applications
## - Data Catalog
## - RESTful Web Service
## - REST Services
## - Scheduler Web Admin Tool
## - Design Studio
## - Diagnostic & Monitoring Tool
frontend http-in
      bind              *:9090
      default_backend backend_servers_tomcat_http
      option                  forwardfor

backend backend_servers_tomcat_http
      balance      roundrobin
      cookie       SERVERID  insert
      server             node01 192.168.141.121:9090 cookie ck_node01 check
      server           node02 192.168.141.122:9090 cookie ck_node02 check

## https/9443 => Proxy for Apache Tomcat-based Web Applications using
## a secure (SSL/TLS) channel
frontend https-in
      bind              *:9443                    ssl                    crt
/etc/haproxy/denodo_server_key_store.pem
      default_backend   backend_servers_tomcat_https

backend backend_servers_tomcat_https
```

```
      balance          roundrobin
      cookie SERVERID  insert
      server           node01 192.168.141.121:9090 cookie ck_node01 check
      server           node02 192.168.141.122:9090 cookie ck_node02 check


## tcp/8000 => Proxy for Scheduler Server traffic
frontend tcp-in-scheduler
      mode                  tcp
      bind                  *:8000
      default_backend         backend_servers_tcp_in_sched
    option              tcplog

backend backend_servers_tcp_in_sched
      mode            tcp
      balance         roundrobin
      server          node01 192.168.141.121:8000 check
      server          node02 192.168.141.122:8000 check


## tcp/9999 => Proxy for JDBC traffic
frontend jdbc-in
      mode              tcp
      bind              *:9999
      default_backend   backend_jdbc_servers
      option                  tcplog

backend backend_jdbc_servers
      mode              tcp
      balance     roundrobin
# These names must match the names defined in the Solution Manager
      server            vdp.node01 192.168.141.121:9999 check
      server            vdp.node02 192.168.141.122:9999 check


## tcp/9997 => Proxy for JDBC traffic (RMI Factory Port)
## The RMI Factory Port (by default 9997) is not needed anymore in Denodo 8.
## If you are running Denodo 7, this must be configured only if the clients
## cannot access directly the cluster nodes. The RMI factory port must
## not be balanced and it must be different for every node.

## Comment this out if running Denodo 8
frontend jdbc-in-factory-node01
    mode                        tcp
    bind                        *:9997
    default_backend             backend_jdbc_srv_factory_node01
    option                      tcplog

## Comment this out if running Denodo 8
backend backend_jdbc_srv_factory_node01
    mode                tcp
    server              node01 192.168.141.121:9997 check

## Comment this out if running Denodo 8
```

```
## The RMI factory port must be different for every node
frontend jdbc-in-factory-node02
    mode                tcp
    bind                *:9995
    default_backend     backend_jdbc_srv_factory_node02
    option              tcplog

## Comment this out if running Denodo 8
backend backend_jdbc_srv_factory_node02
    mode                tcp
    server              node02 192.168.141.122:9995 check


## tcp/9996 => Proxy for ODBC traffic
frontend odbc-in
      mode        tcp
      bind        *:9996
      default_backend backend_odbc_servers
      option            tcplog

backend backend_odbc_servers
      mode        tcp
      balance     roundrobin
      server            node01 192.168.141.121:9996 check
      server            node02 192.168.141.122:9996 check
```

- The global section instructs about process-wide parameters. In this example, we are telling haproxy to use the `rsyslogd` daemon for logging and specifying the user and group of the process.
- The defaults section lists some default parameter values.
- Finally comes the proxies section, one for each open port we want to make available through the load balancer. Each proxy consists in a frontend section and a backend section.
    - The frontend section has to specify
        - the protocol (tcp, http, …)
        - the listening IP address and the port (ex. *:9999 indicates that we want to listen on all the available IP address of the machine)
        - the default_backend
        - And any additional option
    - The backend section has to specify the list of servers in the backend with the following syntax

      `server` `<name> <address>[:[port]] [param*]`

      Every backend section name should match one and only one default_backend name in the configuration. Address can be a hostname (for example defined in `/etc/hosts`) but it is resolved at haproxy service start time.
- A note regarding the SSL/TLS configuration of the load balancer. If traffic between client applications and the Denodo servers is secured through SSL/TLS you should choose how the load balancer deals with secure connections. There are several techniques:

  - **SSL Termination**: client application-to-loadbalancer is SSL/TLS secured, loadbalancer-to-backend is not secured
  - **SSL Pass-through**: the load balancer does not modify the connection, it just proxies it.
  - **Hybrid**: combination of Termination and Pass-through

  Each approach brings benefits and drawbacks, see the article linked in the reference section for an in-depth discussion and implementation tips in HAProxy. In our case we chose SSL Termination.
- A note regarding the RMI factory port (by default 9997) that applies only to versions prior to Denodo 8. This is the port through which communication flows back and forth between client applications and the Virtual Dataport server once the connection is established. If the cluster nodes are not visible from the client applications, i.e. when all the communications must flow through the load balancer, we have to instruct the load balancer to redirect the communication to the proper backend node. Further configuration of the RMI parameters and the hostnames of the backend nodes is needed: instructions can be found in section "Virtual Server and Ports" of the Denodo Platform Cluster Architecture article.

5.3 HTTP SESSION STICKINESS

When behind the load balancer we put web applications that serve different content based on logged-in users, we must find a way to avoid same-session switching between nodes. One of the methods to deal with this is to implement the so-called *application layer persistence*, in which the load balancer inserts a `Set-Cookie:` header that is included in every following request from the same client.

You enable the persistence by adding the `cookie NAME insert` instruction in the backend section along with the `cookie` keyword followed by a unique cookie name in each `server` instruction.

For example:

```
cookie SERVERID  insert
server node01 192.168.141.121:9090 cookie ck_node01 check
server node02 192.168.141.122:9090 cookie ck_node02 check
```

In this case we are associating each newly routed connection to `node01` the cookie `ck_node01`. The next time the same client sends a request to the load balancer it will include the cookie and the load balancer will force the routing to `node01`, thus bypassing the load-balancing algorithm. The same behavior applies for `node02`.

5.4 CONFIGURATION OF RUNTIME API

In a later section we will see how to setup Deployment Scripts in the Solution Manager. To do that we must allow runtime modifications to the HAProxy configuration, for example to cut or enable access to a given backend server.

It turns out that we can do that by using the HAProxy Runtime API (link in the reference section).

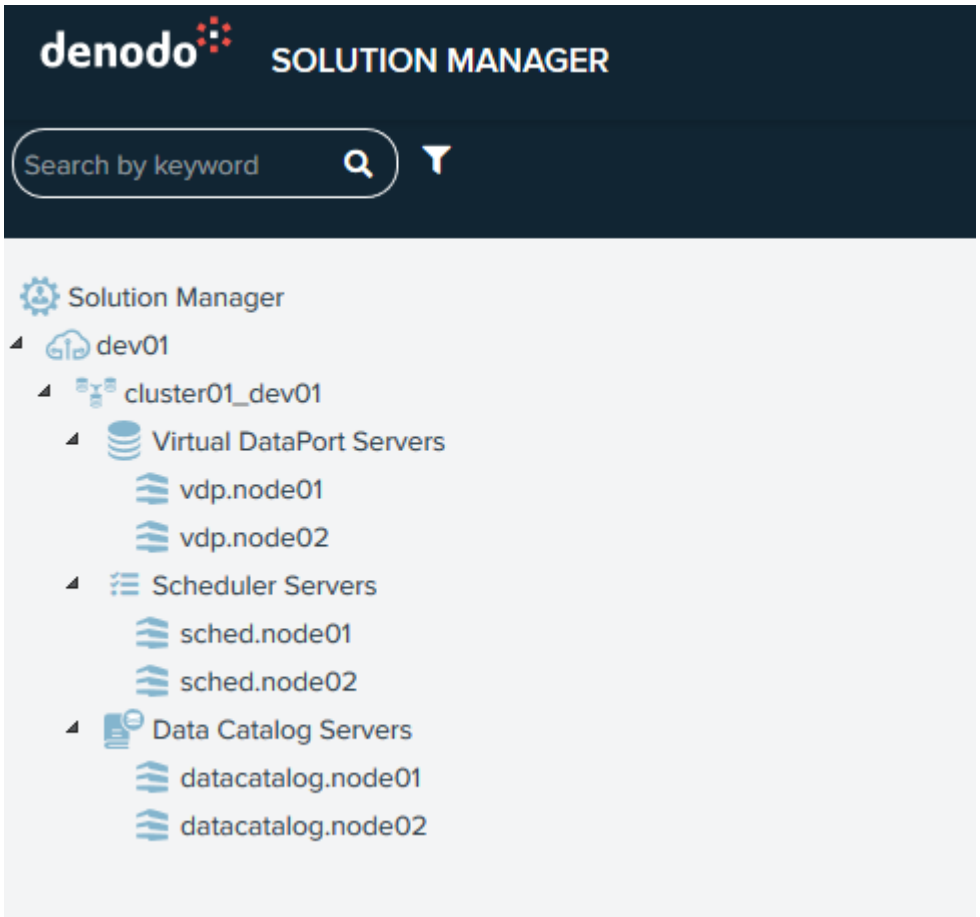
For that to work we just need to add the following lines in the `global` section of the configuration file:

```
stats socket ipv4@127.0.0.1:9999 level admin

stats socket /var/run/haproxy.sock mode 666 level admin

stats timeout 2m
```

6 SETUP OF THE DEPLOYMENT SCRIPTS IN THE SOLUTION MANAGER

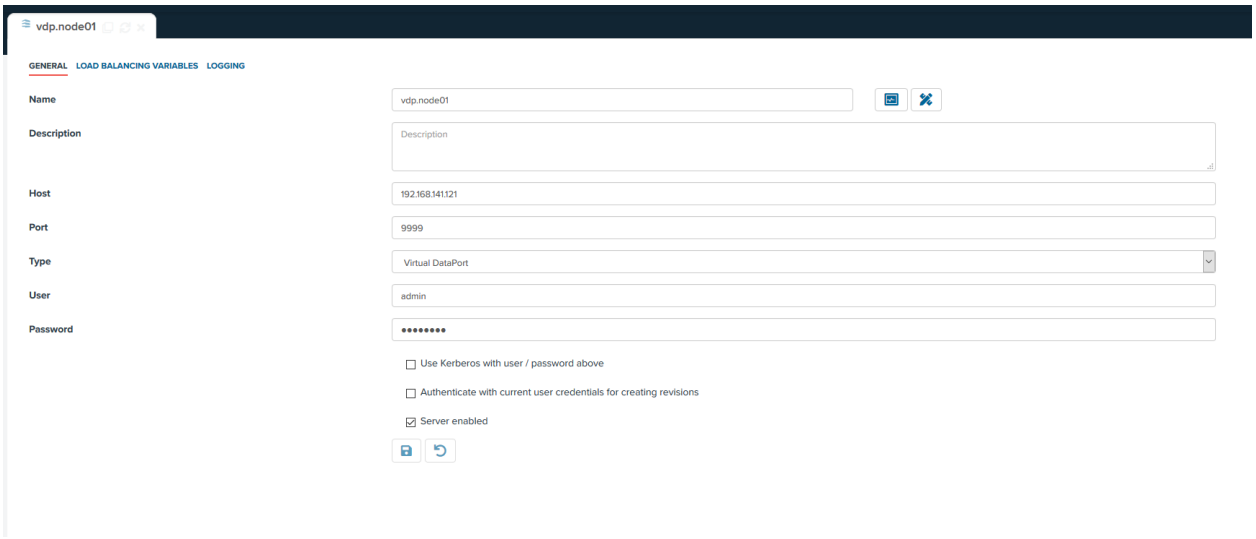
As a preliminary step of the deployment scripts configuration in the Solution Manager you should define:

- an environment (here `dev01`)
- a cluster (here `cluster01_dev01`) in this environment
- a Scheduler, Virtual DataPort and Data Catalog (the latter is compulsory in Denodo 8) servers of each node in the cluster.



Environment, cluster and servers definition in the Solution Manager

As an example, here is the definition of `vdp.node01`:



Definition of a VDP server in the Solution Manager

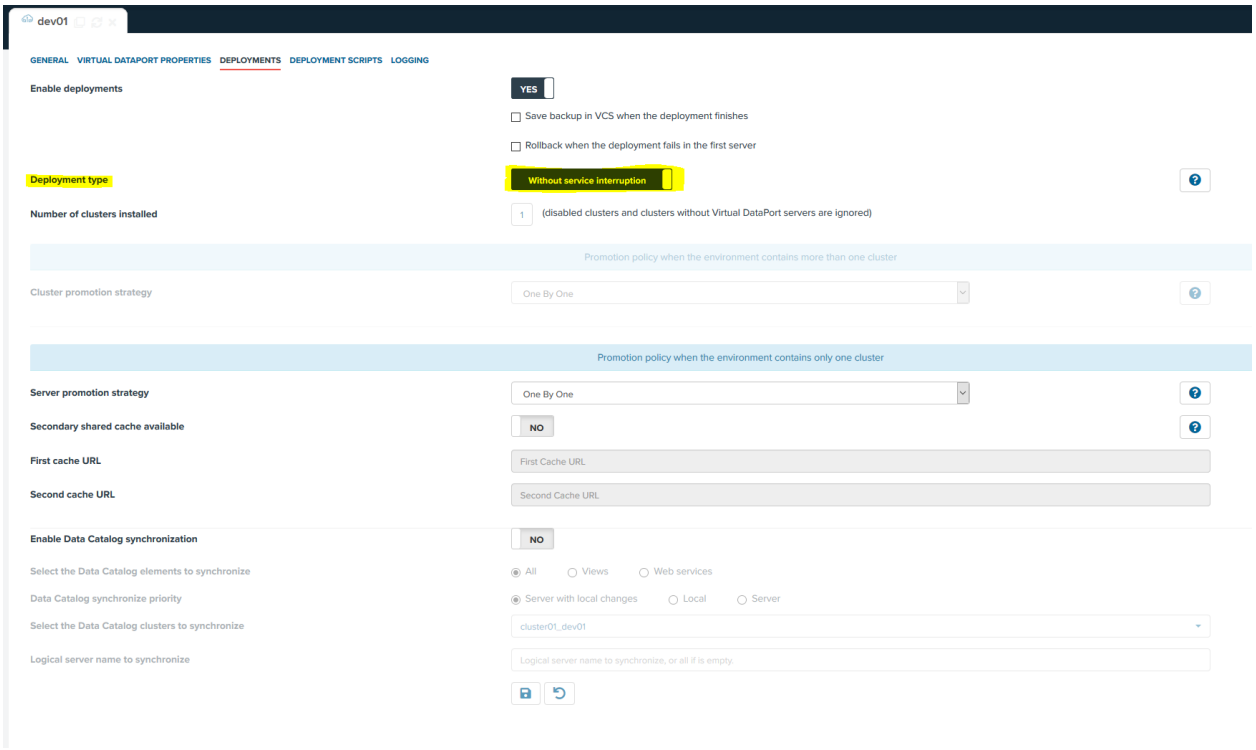
6.1

6.2 DEPLOYMENT SCRIPTS

In the Solution Manager we can define scripts that disable the servers in a cluster before a promotion deployment and bring them back online once the deployment is finished. The idea of these scripts is to communicate to the load balancer to temporarily cut communication to the backend node being updated.

To do that, we need to apply two settings:

1. Define the *Deployment type* as *Without service interruption* . We apply this setting in the DEPLOYMENTS pane of the environment, as shown in the image below:



Setting the Deployment Type in the Environment

2. Upload the deployment scripts in the DEPLOYMENT SCRIPT pane of the environment. We can either cut access at the cluster and node level, in our example we chose the latter.

   NOTE: to get the script working, ensure that
   - **The `sshpass` package is installed in the machine hosting the Solution Manager Server.**
   - **The `socat` package is installed in the machine hosting the load balancer.**
   - **The host public key of the loadbalancer machine is known in the Solution Manager Server machine. This means that an entry with that hostname must be present in file `~/.ssh/known_hosts`. You can do that with:**
     `$ ssh-keyscan -H loadbalancer >> ~/.ssh/known_hosts`

   Here is the script to be uploaded:

```
#!/bin/bash

######################
## This script allows enabling or disabling a node on a
## HAProxy based load balancer
######################
nodename=$1
status=$2
loadbalancer_user=$3
loadbalancer_pwd=$4
```

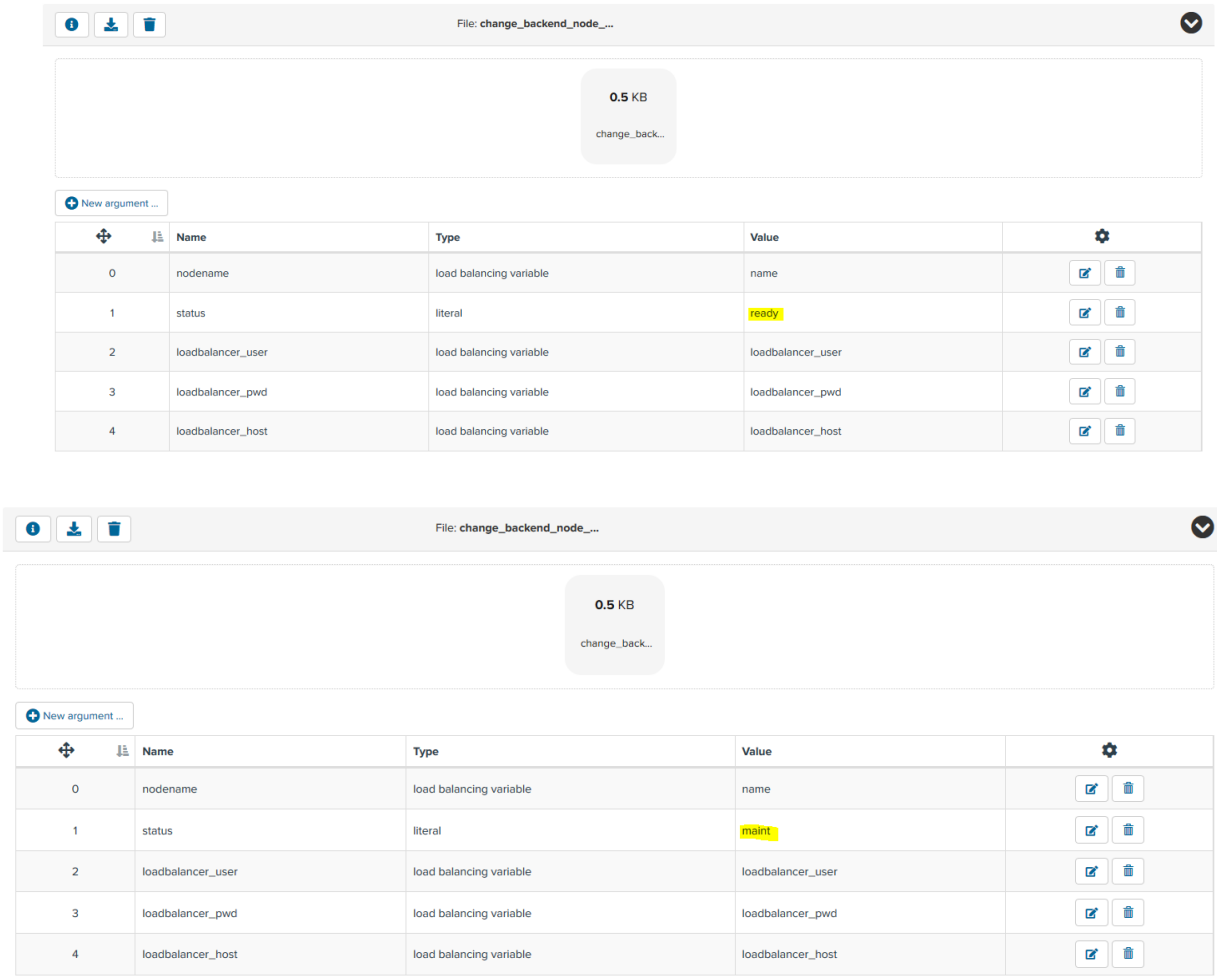
```
loadbalancer_host=$5

set_instruction="set server backend_jdbc_servers/${nodename} state ${status}"

echo "echo ${set_instruction} | socat stdio /var/run/haproxy.sock" | sshpass -p${loadbalancer_pwd} ssh ${loadbalancer_user}@${loadbalancer_host} 'bash -'
echo "EXIT CODE=$?"
```

In this script we are telling HAProxy via a remote SSH command (`sshpass`) that we want to change the status of `backend_jdbc_servers/${nodename}` to `${status}`. As such this script can be used both for enabling (`state: ready`) and disabling (`state: maint`) the relevant node. In this example we only deal with the Virtual DataPort JDBC traffic; in a more realistic scenario you would need to take into account all the components that should be disabled/enabled (e.g. Scheduler and Data Catalog).

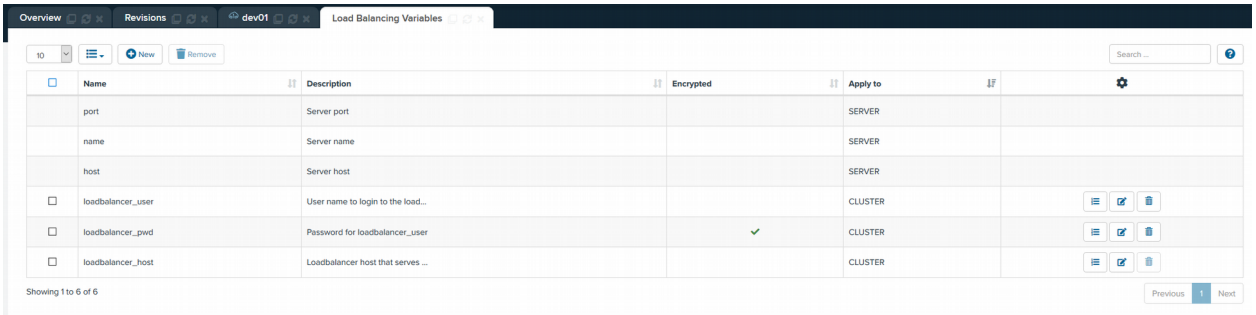
Parameter passing is done by graphically defining parameters. In our example there are five of them:

param_number	name	type	value
0	nodename	load balancing variable	name
1	state	literal	ready
2	loadbalancer_user	load balancing variable	loadbalancer_user
3	loadbalancer_pwd	load balancing variable	**********
4	loadbalancer_host	load balancing variable	loadbalancer_host

Parameters can be of type *literal*, where you specify the value at creation time, or *load balancing variable*, that are defined at the cluster or the server level and can be handy when a parameter corresponds to a node (`nodename`) or to a cluster (such as `loadbalancer_user`, `loadbalancer_pwd` and `loadbalanceer_host`). Sensitive parameters, such as passwords, can be stored encrypted.



Deployment Scripts configuration: enabling (top) and disabling the node (bottom)



Definition of Load Balancing variables at the server and the cluster level

6.3 TESTING OF DEPLOYMENT SCRIPTS

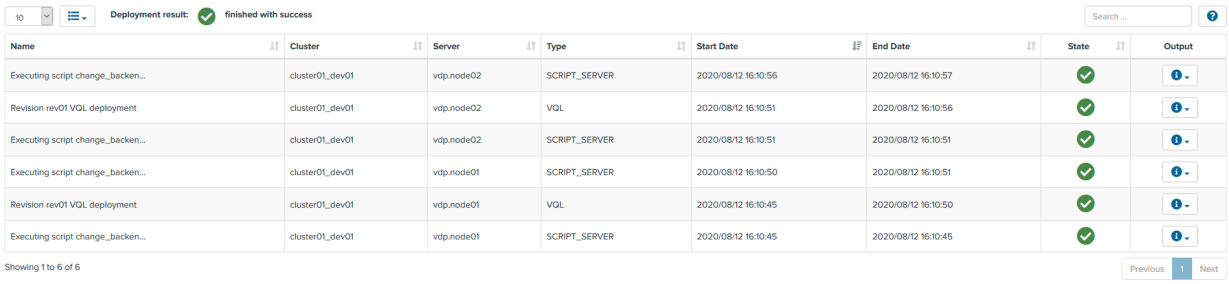
We can test that your deployments scripts work as expected by deploying a revision in the Solution Manager.

Afterwards, we can inspect the load balancer logs (`/var/log/haproxy.log`)

```
Aug 12 16:10:45 localhost.localdomain haproxy[55119]: Server
backend_jdbc_servers/vdp.node01 is going DOWN for maintenance. 1 active and
0 backup servers left. 0 sessions active, 0 requeued, 0 remaining in queue.
Aug 12 16:10:45 loadbalancer haproxy[55119]: [WARNING] 224/161045 (55119) :
Server backend_jdbc_servers/vdp.node01 is going DOWN for maintenance. 1
active and 0 backup servers left. 0 sessions active, 0 requeued, 0 remaining
in queue.
Aug 12 16:10:51 localhost.localdomain haproxy[55119]: Server
backend_jdbc_servers/vdp.node01 is UP/READY (leaving forced maintenance).
Aug 12 16:10:51 loadbalancer haproxy[55119]: [WARNING] 224/161051 (55119) :
Server backend_jdbc_servers/vdp.node01 is UP/READY (leaving forced
maintenance).
Aug 12 16:10:51 localhost.localdomain haproxy[55119]: Server
backend_jdbc_servers/vdp.node02 is going DOWN for maintenance. 1 active and
0 backup servers left. 0 sessions active, 0 requeued, 0 remaining in queue.
Aug 12 16:10:51 loadbalancer haproxy[55119]: [WARNING] 224/161051 (55119) :
Server backend_jdbc_servers/vdp.node02 is going DOWN for maintenance. 1
active and 0 backup servers left. 0 sessions active, 0 requeued, 0 remaining
in queue.
Aug 12 16:10:56 localhost.localdomain haproxy[55119]: Server
backend_jdbc_servers/vdp.node02 is UP/READY (leaving forced maintenance).
Aug 12 16:10:56 loadbalancer haproxy[55119]: [WARNING] 224/161056 (55119) :
Server backend_jdbc_servers/vdp.node02 is UP/READY (leaving forced
maintenance).
```

From the logs, we clearly see that the node access has been cut off and restored one after another.

We can also have a look at the deployment summary in the Solution Manager Administration Tool that confirms that VQL deployments have been bracketed among SCRIPT_SERVER tasks that correspond to script executions in the load balancer.



Deployment status showing that the enable and disable scripts have been run before and after the VQL deployment

6.4 CONSIDERATIONS ON THE DENODO INTERNAL METADATA CATALOGS IN CLUSTERED DEPLOYMENTS

6.5 EACH DENODO COMPONENT MAINTAINS A DEDICATED METADATA CATALOG. WHEN DEPLOYING IN CLUSTERS THE FOLLOWING ASPECTS SHOULD BE TAKEN INTO ACCOUNT FOR THE ARCHITECTURE DESIGN:

6.6 SCHEDULER

In cluster mode, the Scheduler servers that constitute the cluster must share the Scheduler internal metadata database. You can consult the Scheduler Cluster Settings manual page to set up this scenario and to understand the technical implications of having such deployment.

6.7 DATA CATALOG

In cluster mode, the Data Catalog servers that constitute the cluster must share the Data Catalog internal metadata database. You can consult the Data Catalog External Database Setup manual page for the configuration steps.

6.8 VIRTUAL DATAPORT

In cluster mode, the Virtual Data Port servers

- may share the internal metadata catalog starting with Denodo 8
- don't share the internal metadata catalog until Denodo 7 included.

So, in Denodo 7 and in Denodo 8, if each node interacts (read/write) only with its local metadata catalog, there must be an external agent that guarantees the synchronization between all the local metadata catalogs, one for each node.

If, in Denodo 8, a shared metadata catalog is configured, the replication is done automatically.

By default the Virtual DataPort metadata catalog is not shared. To understand how the metadata catalog can be shared and to read some considerations and insight on this choice you can refer to the Storing Catalog on External Database manual page.

Diagnostic & Monitoring Tool

The Diagnostic & Monitoring Tool does not support a shared metadata catalog so the configuration, environments and servers must be synchronized manually.

Summary

Denodo Platform Component	Support for shared DB for metadata	Compulsory shared DB in clusters

Virtual DataPort	yes	no
Scheduler	yes	yes
Data Catalog	yes	yes
Diagnostic & Monitoring Tool	no	no

Summary of support for shared metadata catalogs of the Denodo components

7 RMI SETUP IN DENODO NODES

Note: You need to perform these steps only if using Denodo 7.

For each node in your cluster, apply the following modifications.

1. Change the hostname

```
$ sudo hostnamectl set-hostname node01.ubuntu1804-denodo7
$ hostname
node01.ubuntu1804-denodo7
```

2. Change the RMI hostname in the Denodo configuration files

```
$ grep -R node01 $DENODO_HOME/conf
conf/scheduler/ConfigurationParameters.properties:Launcher/registryURL=node0
1.ubuntu1804-denodo7
conf/vdp/VDBConfiguration.properties:com.denodo.vdb.vdbinterface.server.VDBM
anagerImpl.registryURL=node01.ubuntu1804-denodo7
```

3. Change the RMI factory port in the Denodo configuration files (only Virtual DataPort):

```
$ grep factoryPort $DENODO_HOME/conf/vdp/VDBConfiguration.properties
com.denodo.vdb.vdbinterface.server.VDBManagerImpl.factoryPort=9997
```

   This port must be different for each Virtual DataPort node behind the load balancer and its value must be used in the corresponding directive in `haproxy.cfg`.

4. Adjust the RMI factory port in the load balancer configuration. See for example `backend_jdbc_srv_factory_node01` and `backend_jdbc_srv_factory_node02` in the configuration file shown above.

5. Check that the license is provided via the Solution Manager:

```
$ grep license.host $DENODO_HOME/conf/SolutionManager.properties
conf/SolutionManager.properties:com.denodo.license.host=ubuntu1804-den7-sol-
man
```

6. Regenerate the shell files and restart the servers (note that if you did not define the node in the Solution Manager, the node will refuse to start)

```
$ $DENODO_HOME/bin/regenerateFiles.sh
```

```
$ $DENODO_HOME/bin/vqlserver_shutdown.sh
$ $DENODO_HOME/bin/vqlserver_startup.sh

$ $DENODO_HOME/bin/scheduler_shutdown.sh
$ $DENODO_HOME/bin/scheduler_startup.sh

$ $DENODO_HOME/bin/datacatalog_shutdown.sh
$ $DENODO_HOME/bin/datacatalog_startup.sh
```

8 TESTING

You can test your Denodo clustered deployment setup by:

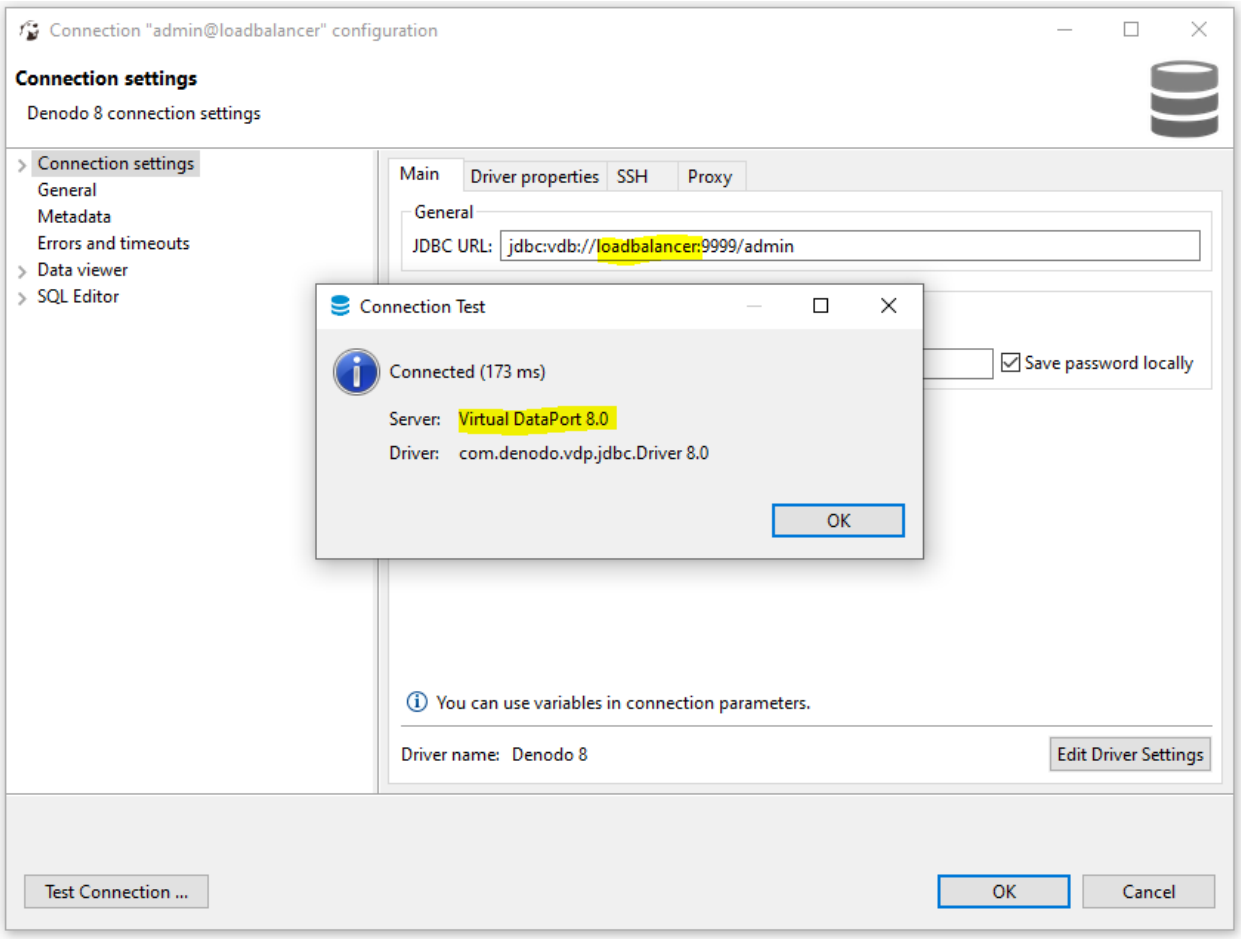
- trying to reach all the Denodo services through the load balancer.
- verifying that the load is balanced by checking that the nodes receive the expected portion of a simulated workload, taking into account the workload distribution algorithm logic. To this end, you may find it useful to monitor both the Denodo nodes via the Diagnostic & Monitoring Toll and the logs of the load balancer.

8.1 WEB BASED APPLICATIONS

- **Data Catalog**: Login, do some typical actions (search, query, categorize, administer..) then logout. Repeat.
  - HTTP: http://loadbalancer:9090/denodo-data-catalog/AutoLogin
  - HTTPS: https://loadbalancer:9443/denodo-data-catalog/AutoLogin
- **Scheduler Web Admin**: Login, do some typical actions (explore jobs, create a new one, delete one job, edit one job, ...) then logout. Repeat.
  - HTTP: http://loadbalancer:9090/webadmin/denodo-scheduler-admin/login
  - HTTPS: https://loadbalancer:9443/webadmin/denodo-scheduler-admin/login
- **Design Studio (only Denodo 8)**: Login, do some typical actions (explore catalog, open a view, create a derived view, execute it, ..) then logout. Repeat.
  - HTTP: https://loadbalancer:9090/denodo-design-studio/#/
  - HTTPS: https://loadbalancer:9443/denodo-design-studio/#/
- **RESTful Web Service**: Login and do some typical actions (navigate views, search with url parameters, ...) then logout. Repeat.
  - HTTP: http://loadbalancer:9090/denodo-restfulws/
  - HTTPS: https://loadbalancer:9443/denodo-restfulws/
- **REST Web Services**: Query a deployed REST web service
  - HTTP: http://loadbalancer:9090/server/<virtual database name>/<web service name/views/<view name>?param_1=val_1&param_2=val_2[...]
  - HTTPS: https://loadbalancer:9443/server/<virtual database name>/<web service name/views/<view name>?param_1=val_1&param_2=val_2[...]
- **Diagnostic & Monitoring Tool**: Login and do some typical actions (create a new monitored environment, refresh the requests table, ...) then logout. Repeat.
  - HTTP: http://loadbalancer:9090/diagnostic-monitoring-tool/Home
  - HTTPS: https://loadbalancer:9443/diagnostic-monitoring-tool/Home

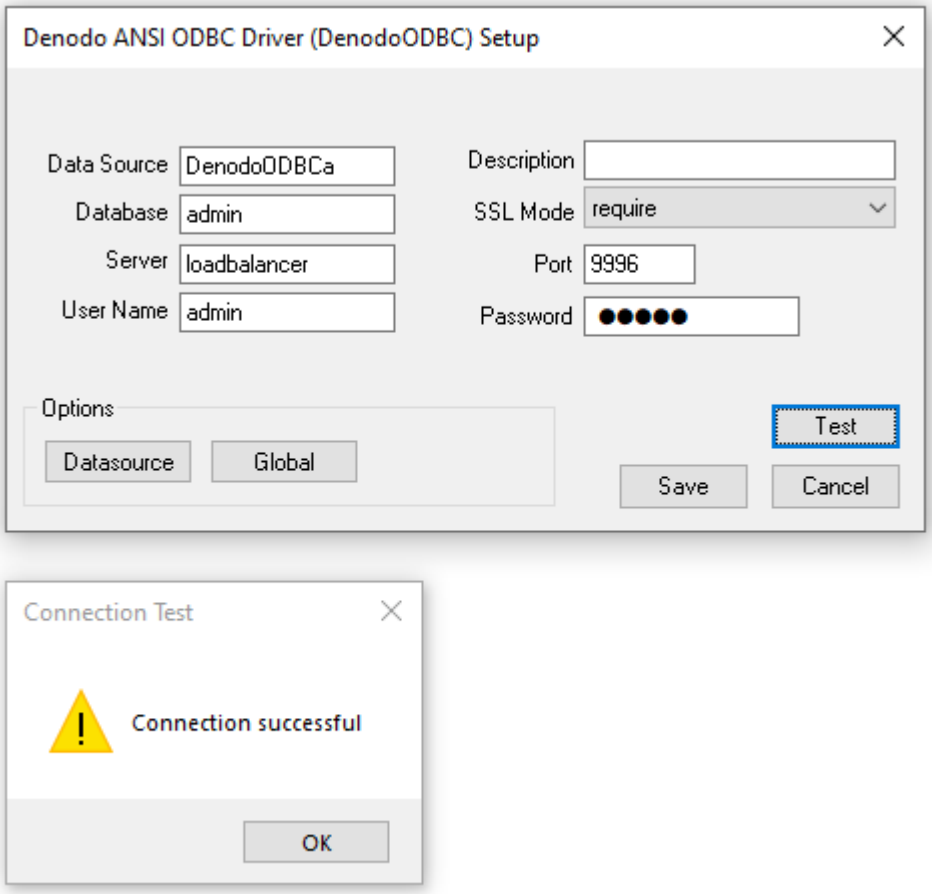
8.2 JDBC CLIENT

(ex. DBeaver 7.1.4)



Successful connection to Virtual DataPort using JDBC

8.3 ODBC CLIENT



Windows ODBC Data Source Creation

8.4 SCHEDULER

You have several options to test the configuration:

- **Graphically**: Connect to the scheduler server using the Scheduler Web Administration Tool specifying the loadbalancer hostname
- **Command line**: Customize the sample script provided under `$DENODO_HOME/samples/scheduler/scheduler-api/` that leverages the Scheduler RMI Client API. You will probably have to modify the parameters in `samples/scheduler/scheduler-api/src/com/denodo/scheduler/demo/GlobalNames.java`, launch `$DENODO_HOME/samples/scheduler/scheduler-api/scripts/test_schedulerclient.sh`. After these steps, you can run a job from the command line:

```
$DENODO_HOME/samples/scheduler/scheduler-api/scripts/test_schedulerclient.sh
-start job_extract_iv_sales -h loadbalancer -p 8000
```

- **Command line (Denodo 8)**: Leverage the Scheduler REST Client API

9

10 REFERENCES

HAProxy in Wikipedia
HAProxy Home page
Dynamic Configuration of HAProxy via API
HTTP Session stickiness
HAProxy tutorial
Using SSL Certificates with HAProxy
Configuring Deployment Scripts in the Solution Manager