Denodo Platform Cluster Architecture

Revision 20220311

Denodo Inc.: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA www.denodo.com

CONTENTS

1 OVERVIEW

A group of Denodo servers sharing the same metadata can be clustered to offer High Availability (HA), scalability and improved performance. In essence, a Load Balancer presents a "virtual server" address to the outside world. When users attempt to connect, it forwards the connection to an available server, taking care of bi-directional network address translation in doing so.

1.1 THE LOAD BALANCING PROCESS

Generally speaking, a basic load-balancing transaction works as follows:

1. A client attempts to connect with a service on the load balancer using its configured virtual server IP and port number.
2. The load balancer accepts the connection, and, after deciding which host should receive the connection, changes the destination IP (and possibly port) to match the service of the selected host.
3. The host accepts the connection and responds with data packets back to the original source, the client, via the load balancer.
4. The load balancer intercepts the returning data packets from the host, changing their source IP headers (and possibly port headers as well) to match the original virtual server IP and port, and forwards packets back to the client.
5. The client receives the returning packets, as if they had come directly from the virtual server.

1.2 RECOMMENDED ARCHITECTURE

The following elements are important to consider in determining how best to configure optimal performance and evenly-distributed load in a Denodo environment:

A. **Load Balancer** – The load balancer will route new connections to a healthy node of the Denodo cluster, distributing the load in an even way among the available servers.
B. **Denodo Servers ("nodes")** – The execution of each query will be done on one of the Denodo servers belonging to the cluster.
C. **Connection Pooling** – Client applications can establish connection pools to Denodo, increasing performance and better ensuring healthy connections.
D. **Cache** – In a clustered configuration, the cache can be shared by all the nodes of the cluster, enabling any node in the cluster to take advantage of data previously cached by any other node in the cluster.
E. **Web Services** – The Denodo server launches an embedded Apache Tomcat web container, where published web services may be "internally" deployed. In this case, the load balancer should also be configured to distribute HTTP requests made to the web server port (9090 by default). Denodo web services can also be exported and deployed in any external Java application container. In that case, the same balancing considerations pertain as when Denodo's companion web container is being used.

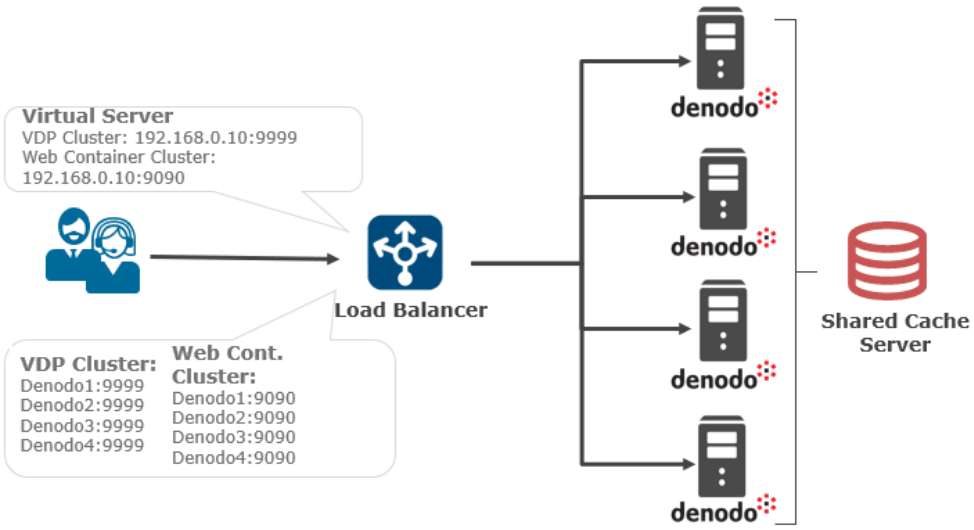

Figure 1: Denodo Cluster Setup

In a typical configuration like the one seen above, the load balancer contains a list of participating member nodes. Since a node is defined as a combination of IP and port numbers, clusters for JDBC (RMI Registry port 9999) as well as for web service access (HTTP port 9090) are defined. The client applications are now pointing to the load balancer address, using it as a “virtual server”. When connections are created, the load balancer routes them to a specific member of the cluster following whatever balancing strategy has been configured on it.
In the following section, we describe configuration and other considerations to keep in mind during the design and operation of a cluster of Denodo servers.

2 CLUSTER CONSIDERATIONS

2.1 LOAD BALANCER

Denodo has been used in production clusters both with hardware and software load balancers, such as:

- Big IP F5
- Cisco ACE
- HP Service Guard
- HAProxy
- Linux Virtual Server
- Amazon ELB

The configuration of a Denodo cluster in the load balancer should be a straightforward process as shown in Figure 1 above.

2.1.1 Virtual Server and Ports

The list of nodes for a virtual server in the load balancer is configured based on server and port. Since Denodo uses different ports for RMI, ODBC and ADO.NET, and HTTP, we will need to configure three virtual servers. The default ports for the Denodo platform are:

- RMI Registry (JDBC): 9999
- ODBC and ADO.NET: 9996
- HTTP: 9090

Some recommendations when using the HAProxy/ELB.
For Denodo 8:

1. Each VDP server node must resolve the RMI hostname to their own IP by editing the `/etc/hosts` file of each VDP server host.

For versions previous to Denodo 8:

1. The RMI hostname of all the VDP Server nodes should be the same as defined in the Load Balancer.
2. Each VDP server should use a different port number for the RMI Factory port (by default, 9997). The RMI Factory ports have to be configured in the balancer too and they must redirect to the right VDP server (i.e. these ports must not be balanced, but load balancer must redirect each one of them to the appropriate VDP server).

2.1.2 Firewall considerations

With Denodo 8.0, firewall configuration for the JDBC interface is made simple. The JDBC (along with the administration tool and the new Design Studio) connections now uses only one port to communicate with Virtual DataPort Server (by default, 9999). In previous versions, the communication requires two ports (9999 and 9997). This change simplifies the configuration of firewalls.

Please note: ODBC, ADO.NET and HTTP connections do not use this port and

therefore this consideration does not apply if those are the only access methods in use.

2.1.3 Server affinity

Although Denodo Servers are stateless (the result of a transaction will not depend on previous executions) the "server affinity" needs to be configured in the balancing process because of how the RMI protocol works. See Virtual DataPort Connectivity for more information.

2.1.4 Balancing strategy

Round Robin strategies tend to be the most effective in scenarios involving queries of similar “weight” (measured as a combination of response data per query to the sources and post-processing in Denodo), and where the nodes are equally powerful.

In mixed scenarios where “heavy” informational queries coexist with “lighter” operational queries, or when the technical specs of the nodes are different, round robin can lead to node overloads (see **FAQ** for more details). More complex load information can be gathered from the nodes and used for advanced status analyses (memory usage, CPU load, etc.). Load balancers usually have strategies that are able to gather and use this information to provide more intelligent balancing decisions.

For example, BigIP F5 offers a strategy called “**Dynamic Ratio**” to cover those scenarios (see appendix for details on how to configure this balancing strategy).

2.1.5 Health check

Denodo provides a health check script that checks a server to ensure it is capable of responding to queries. It can also execute a sample query and advertise the number of results returned, if necessary.
Most load balancers can use external scripts to check the status of member nodes (BigIP F5 refers to these as “External Monitors”). If this capability is not available, a ping to a node’s port can be used as a substitute. This option, although faster, provides a less reliable health check than the Denodo script.
For more information, refer to the section Connecting from a JDBC Client Through a Load Balancer and Using the Ping Script fn the Virtual DataPort Administration Guide.

2.1.6 User Agent

Some load balancers replace the origin IP address of the client applications in the requests sent to the Virtual DataPort Server, in those cases, the property userAgent can be used to retrieve the IP of the original client application.

2.2 CONNECTION POOLS AND DRIVER CONFIGURATIONS

Client connections to the Denodo Server are normally managed by a connection pool. Initializing a new connection in a pool is costly, and in many cases unnecessary. With connection pooling, each newly-created connection is placed in the pool and can be reused, reducing the amount of new connections that will need to be generated. However, the use of connection pools with inadequate balancing strategies, in certain scenarios, can lead to uneven load distribution. You can find a more detailed discussion in the FAQ “Should I use servers with different specs as nodes of my

cluster?".

Another important aspect of connection pools is their ability to check the health of a connection with a ping query. In the context of a cluster, if a ping query fails, the pool considers the connection unusable and tries to obtain another one. Since different connections point to different servers from the same pool, this mechanism can always expose healthy connections to clients even when one or more servers go down.

There are a few connection properties that have to be set in the driver to take advantage of these clustering capabilities. These properties are fully documented in the Virtual DataPort Developer Guide, section "Access through JDBC".

2.2.1 Socket reuse

For JDBC connections and **for Denodo 7.0 only**, set the property "`reuseRegistrySocket`" to "`false`". RMI-based connections will be more evenly distributed across the nodes of the cluster with this configuration.

2.2.2 Ping query timeout

Some connection pools do not support ping query timeouts (which determines how long to wait for a response before a connection is invalidated). The Denodo JDBC driver provides a "`pingQueryTimeOut`" property for these connection pools.

For example: if the URI of the JDBC driver has the properties

```
pingQuery=SELECT * FROM dual()&pingQueryTimeOut=500
```

then every time the pool executes the ping query, the driver will only wait 0.5 seconds for a response (`pingQueryTimeOut` is in milliseconds). Otherwise, the driver throws a SQLException because the TimeOut has been reached, and the pool will consider the connection invalid.

2.3 CACHE

In the context of a Denodo cluster, the Virtual DataPort server cache can be configured in two different ways:

- As a single, shared cache database; accessible to all the nodes of the Denodo Cluster.
- As multiple, independent cache databases; one per Denodo node.

In selecting the appropriate architecture for the cache system of a cluster, considerations such as location of the cluster nodes and cache performance become relevant. For example, if you have nodes in China and the United States, you may decide to have two independent caches, one in each location, instead of moving results over a network for large distances.

For more information on how to set up the cache, refer to the Virtual DataPort Administration Guide, section "Configuring the Cache".

2.3.1 Shared cache

In most deployments, a single, shared cache server is typically recommended:

- A single, shared cache server will provide a higher cache hit ratio – any node can immediately reuse any previously cached result set.
- Reduced IT maintenance – There is only one instance to be set up.
- Less hardware.

However, there are some considerations to bear in mind:

- Access to the cache can become a bottleneck with too many users. In this case, Denodo provides several options:
    - The cache database itself can be clustered for increased efficiency. Database vendor technology such as Oracle Real Application Clusters can be employed in support of the Denodo cache.
    - Cache Connection pool sizes can be increased.
    - Different virtual databases can point to different cache servers to spread the load of supplying cached results to incoming client queries.
- When the metadata of the cached views is modified in a single node, it will affect all of the other nodes.

2.3.1.1 Cache Maintenance Tasks

When multiple Denodo server nodes are configured, **only one of them should be configured for performing the necessary cache maintenance tasks** (such as deleting expired data from the database). Having all nodes managing these tasks is redundant and will create an unnecessary performance impact on the cache database.

Configure the "`Maintenance Period`" setting with an adequate value in one of the Denodo servers and disable cache maintenance for every other server in the Denodo cluster by setting "`Maintenance`" to "`Off`" in the "`General Settings`" tab of the cache configuration.

2.3.2 Independent caches

In clustered environments where Denodo is running in multiple physical locations, a shared cache can introduce increased cache hit latency. Locally-deployed cache databases can provide faster local access.
To enable this architecture, simply use different cache servers in each node's configuration.

2.3.3 Cache replication

When deploying independent Denodo cache databases as described above, it may be a good idea to replicate cached data from a designated "master node". This simplifies the cache preloading process and reduces execution times if the original data sources are local to one of the nodes but remote to others.
When a use case or scenario involves only controlled cache pre-loads, the best option is to rely on vendor-specific database replication techniques. The cache loading process can be scheduled on the "master node", and the data will automatically propagate to the other nodes of the cluster. Oracle, MS SQLServer and MySQL all include automatic replication tools that can be set up for this purpose.
When queries to the different Denodo nodes access distinct, non-overlapping data

sets, it may make more sense to use a "`Partial`" type cache. Since each node controls its own insertions automatically, no cache replication should be required in this case.

2.4 METADATA REPLICATION

In order to keep Denodo Cluster nodes in sync, Denodo metadata must be replicated as well.
Denodo 8.0 includes the Solution Manager that provides support for metadata replication in different nodes of a cluster. The Clusters-Configuration and Promotions section of the Solution Manager Administration Guide provides more detailed information about this functionality for creating and managing clusters.
Starting from Denodo 8.0 the metadata can be hosted on a separate RDBMS: Storing the Catalog on an External Database. If you intended to have a shared metadata for the clusters, you should be aware of the following considerations: Storing the Metadata on an External Database - Considerations About This Feature.
If the Virtual DataPort servers of a cluster share the same metadata (hosted in an external database), you only need to perform a change in one of the servers and it will be propagated to the others, automatically.

2.5 MONITORING

2.5.1 The Diagnostic and Monitoring Tool

The Denodo Platform, starting from version 6.0, includes the Diagnostic and Monitoring Tool. This tool allows monitoring the current state of a Virtual DataPort server or an environment (group of servers). The Diagnostic & Monitoring Tool displays the status of the Virtual DataPort servers using graphs and tables that are updated every few seconds. More information about the monitoring capabilities of this tool can be found in the Monitoring section of the Diagnostic & Monitoring Tool Guide.

2.5.2 JMX

The Denodo Server exposes several internal events using JMX, an industry standard that can be integrated with any cluster monitoring software. Denodo has been successfully monitored from commercial and open source tools such as:

- Nagios
- HP OpenView
- Cacti

The information Denodo exposes includes:

- Overview of memory and CPU usage
- Summary of open and active connections
- Details about current requests and transactions
- Details about cache usage and load processes
- Details about outbound connections and pools for each data source
- Alerts about DDL (metadata modification) operations

2.6 CONCLUSION

Denodo's out-of-the-box capabilities enable well-balanced clustering using commercial load balancers. Denodo clusters can be set up using almost any load balancer on the market, including popular hardware balancing solutions such as Big IP's F5.

3 FREQUENTLY ASKED QUESTIONS

3.1.1 How are requests for data services (non-persistent) vs. persistent connections handled?

Connections to data services are carried out by HTTP requests issued to Denodo-published web services. The web services will manage their own connection pools connecting to a Denodo server node and issue the corresponding queries in an efficient way. In the context of a cluster, the most common configuration has each Denodo exposing data services with its own embedded web services container. The client will request a web service using the load balancer's DNS and the load balancer will route each HTTP request (always stateless in this context) to a node in the cluster. The responding web service will then use its own connection pool to the co-located Denodo server running on the same node to execute the request.
Requests issued via JDBC or ODBC connections will be forwarded through the load balancer which will route the connection directly to one node.
If the client is using a connection pool, a connection can be used for other queries until the connection expires. While the connection remains alive, it will point to the node initially assigned to it by the load balancer at initialization time. Since each connection is balanced at initialization time, it will ensure an even distribution of the connections between the different nodes, and an even distribution of the queries executed using that pool.

3.1.2 If a node in the cluster goes down, what is seen by applications which were connected to the node?

Denodo recommends managing client application connections in a client connection pool. Apart from the performance benefits, client-side connection pools offer the capability of issuing a health check (ping query) before reusing a previously opened connection. If the ping query fails, the connection pool will choose a different, healthy connection (new or reused) pointing to a different node in the cluster. This ensures that every query will be directed to a healthy server.
When a server is removed from the cluster, two scenarios can result:

1. If the server hangs abruptly during the execution of the query, it will be considered an error, and the driver will signal a connection exception back to the client.
2. If the server is purposefully removed from the cluster by a Denodo administrator, for example when upgrading its operating system, the executing queries will finish normally as explained above.

3.1.3 Should I use servers with different specs as nodes of my cluster

No, Denodo recommends all the nodes of the cluster to be servers of similar specs. Heterogeneous deployments may lead to poorly balanced situations, especially when round robin load balancing is used.
To illustrate the problems that may arise if those assumptions are violated, consider a cluster with 2 nodes of different capacity; Node A is faster than Node B. Consider the case of a client application with no connection pool that executes 10 concurrent queries

on the cluster:

1. All 10 queries are executed, 5 go to Node A and 5 to Node B (which is slower than the Node A).
2. All of the queries sent to Node A finish at approximately the same time while the queries on Node B are still running.
3. The client executes another 5 queries that are balanced (3 in Node A and 2 in Node B). At this point, there are 3 queries running in node A and 7 in Node B. Therefore, queries in Node B will be even slower (7 concurrent queries instead of 5) and queries in Node A will be even faster (3 concurrent queries instead of 5)
4. The 3 queries in node A finish, while let's suppose only 1 query in node B was able to finish by now.
5. The client executes yet another 4 queries that are balanced 2 on Node A and and 2 on Node B. At this point, there are 2 queries running in Node A and 8 in Node B. Again, the queries in Node B will be even slower and the queries in Node A will be even faster than ever.

It is easy to see how this process can overload the slower node while wasting the processing power of the faster nodes. The workload is not distributed in the most efficient way with Nodes of different capacities and a round-robin balancing strategy.

There are 2 recommended solutions to the scenario of nodes with different capacities depending on whether we are using a connection pool or not:

- When a connection pooling in use, connections are balanced when the connection is first created and the connections are reused for different queries over time.
  - If a round robin balancing strategy is in place, the situation described above is not very likely to happen. The connection pool will just reuse as many available connections from the faster server that it can in order to execute new queries and not send the queries to each node.
  - Dynamic ratio methods are usually not advisable with connection pools. Since the connections in the pool are reused, the information used for the balancing decision when the connection is established will probably not be relevant in the future. Thus, if Dynamic Ratio is the chosen balancing strategy, connections in the pool must be renewed often so that they can be balanced based on more current conditions.
- If a connection pool is not used, a dynamic ratio method is advisable when the executed queries or the nodes in the pool are heterogeneous. A dynamic ratio can adjust for the workload of each node. For instance, in step 3 of the above example, the 5 new queries should all be assigned to the faster Node A since it is idle, and the slower node would not receive additional queries until the initial batch is finished.

3.1.4 Should I mix different types of workload (i.e. operational and informational) in the same cluster

If the workload of a cluster is very heterogeneous, it could lead to similar situations to the one described in the previous question, except that the queries themselves would take a different amount of time rather than the nodes having different processing

speeds. For example if some queries take 100ms to get executed while others last up to 10 minutes, you may inadvertently send all of the 10 minute queries to 1 node and the 100ms queries to another. One would then be overutilized and the other underutilized.

In general, Denodo recommends having homogeneous clusters and homogeneous queries. If there are two different use cases or applications with very different query workloads, Denodo recommends setting up different clusters for each set so that the workload is more evenly distributed. This also allows more precise set ups tweaked to the specifics of each scenario; memory heap space, for example.

However, if the recommendation is not possible, the same considerations for nodes of different capacity around balancing strategies stated in the question above will apply here.