How to implement web service versioning in Denodo

Revision 20200527

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

1

2 INTRODUCTION

It is common that web service APIs evolve through time. New columns appear and others get deprecated, for example.

When the signature of a web service changes, consuming applications need to be modified accordingly. Coordinated migrations of all applications at once is usually not an option since they are owned by different teams with different life-cycles. Therefore, to reduce the impact on consuming applications, it may be needed to keep the older version available for a period of time, while the transition happens without disrupting the consumers.

In Denodo, it is possible to implement versioning in a couple of different ways explained in this document.

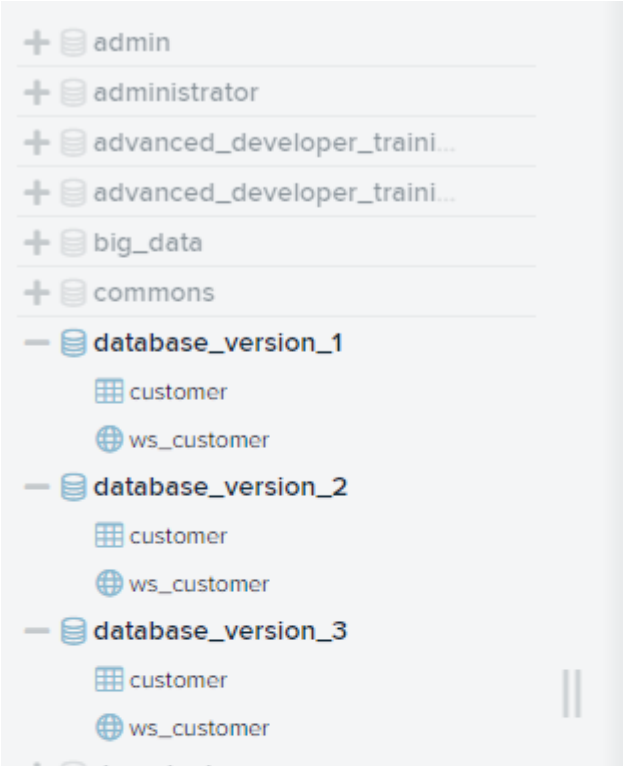
3 MULTIPLE DATABASES AND VERSION CONTROL SYSTEMS

Denodo offers native integration with version control systems like Subversion, Git and Microsoft TFS. Using the integration with version control systems it is possible to keep track of different versions of the same object. The different versions correspond with different revisions of the version control system repository. For example, let's say that there is an original version `v1` of a virtual database, and then some changes are made to the database and version `v2` is committed to the repository.

How to serve `v1` and `v2` simultaneously?
The main idea is to have different databases in Denodo, all of them in sync with the same repository, but that had been checked out to different revisions. When running a checkout to a repository, Denodo shows you a drop-down with the existing revisions to select the one needed.

Let's say there is a virtual database named "`database`", and current revision is 1:

- `databaseA`: this database with no version in its name will represent the latest release. it will always be synchronized with the latest revision.
- `database_version_1`: named version that serves also the latest release.
- `database_version_2`: named version that serves the previous release.



The use of a database without a version name allows us to offer an endpoint that is version independent and will always serve the latest release. The use of databases with version names allows us to "lockdown" the endpoints to a specific version. In most scenarios it is better to keep a limited number of versioned releases available to end-users, in this example only two, to allow them to migrate during the lifetime of that version.

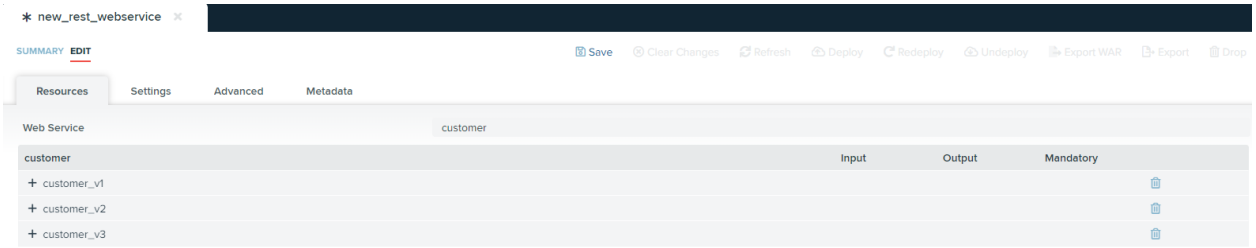
Since the URL of a web service published in Denodo contains the database name in its URL, the endpoint URL will be something like this:

```
http://host:9090/server/database_version_1/myservice/views/customer
```

This approach involves an entire database for the new version. It makes more sense if the migration involves the entire database. For example, if there are global monthly or quarterly updates with changes to several services.
However, this approach does not work if each individual service in a database needs to evolve independently and at any point in time. If that is the situation, the second option will be a better fit.

Versions as different views in the same database

This approach is based on the idea of using different versioned views, all built on top of the same common underlying view. The version can be some naming convention, for example a suffix like `view_vXY`.
Then, the same web service can include different versions:



In this case, the version is in the view name, and the URL will look like:

```
http://host:9090/server/databasea/myservice/views/customer_v1
```

As with the previous approach, it is a good practice to offer a non-versioned version ("`customer`" in the example above) that always matches the latest version.

From a migration perspective, Denodo gives view granularity when exporting a web service. This way it is possible to select the views to include in a VQL export file. For example, it is possible to include a `v11` view only as `v12` is still under development.
NOTE: This feature was included in Denodo 6.0 update 20161219

Since this approach gives individual control for each service and view, it offers greater flexibility. Each service can evolve independently and adapt to changes in the sources and user needs.
However, from the operations point of view, this flexibility increases the load on the migration team and Q&A compared with the periodical version. This should be taken into consideration when considering the staff required to manage the solution.