Data Persistence in Containers

Revision 20221018

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

CONTENTS

1 INTRODUCTION

Containers technology is gaining a lot of popularity due to all the benefits it provides. Their efficiency, portability, and agility are making a revolution in the IT departments of many companies. However, with great power comes great responsibility, and so it is important to learn about how they work before including them in our architecture.

One of the most unexpected facts about containers is that they do not persist the data by default. Their ephemeral nature means that once the container dies and the death container is removed, the data inside the container is also gone. There are some cases where this is not a problem, for instance, if we do not expect changes in our application, or if the Denodo metadata is stored in an external database. On the other hand, if we are using the internal metadata database any time a user, role, view,.... or any other element is created or changed the container will change and we need to consider what files and folders need to be persisted.

In general, we will need to persist data such as Denodo metadata or custom configuration files, so the data remains available once the container dies.

2 PERSISTING DATA IN DENODO CONTAINERS

By default, the data inside the Denodo containers is not persisted. So, if we start a Denodo container and create some views, the views will be lost once the container dies. If those views have to be saved then we will need to do something about them:

1. We can generate a new container image containing the changed data.
2. We can export the views before the container dies, although this does not seem very practical.
3. We can use the Version Control System integration of Virtual DataPort to persist the metadata of our Denodo server.
4. We can mount volumes from the host in the container so the data is persisted there.

2.1 PERSISTING CHANGES IN THE IMAGE

If your denodo deployment will only have changes to its metadata during the deployment of new features then you probably want to persist the changes as part of the image, so you may benefit from the rollout and rollback functionality provided by the container management infrastructure that you could be using, like the Kubernetes deployment mechanisms.

You should also consider that data for some parts of the deployment should reside outside of the Denodo container to start with, like the user authentication database, or the database for the cache if in use.

With Docker you can create a new container image from a running container with the following commands:

```
docker stop <my-container>
docker commit <my-container> <my-image>:<my-tag>
docker start <my-container>
```

The *docker commit* creates a new Docker image with the changes in the container. This will allow you to run new containers with the same status as the container that you used to create the image. Although it is preferable to generate images with Dockerfiles, this is still a valid option to generate images in a development environment that you can later deploy in Production.

2.2 USING VOLUMES TO PERSIST CHANGES

Using volumes is the typical solution to persist data out of the container. Using volumes simplifies the management and is a good solution for development environments.

In Docker, we can add volumes in the *docker run* command with the option *-v*. For instance, in order to persist the metadata folder we can execute the following command:

```
docker run -v "C:/Denodo/Metadata:/opt/denodo/metadata" denodo-platform:8.0-latest ./denodo-container-start.sh --vdpserver
```

So, now the question is to decide which folders from the Denodo installation folder should be persisted. Basically, there are two approaches:

- Persisting the whole `/opt/denodo` folder: if we want to work with Denodo as a classical installation, this option will enable that functionality.
- Selecting specific folders to persist: instead of persisting the whole folder, you can control which folders you want to persist. That way, you still can take a container-approach to work with Denodo, and save some disk.

Hence, if we take the second approach, we will need to select the folders to be persisted. In order to make the decision, we will go folder by folder and check what is being stored there. If the folder contents will not change in our environment, then that folder does not need to be persisted. Only folders whose content might change should be persisted.

For instance, the following folders are subject to change in some scenarios and may require to be persisted:

Folder	Reason
`/opt/denodo/bin`	The Denodo scripts can be regenerated after changing the JVM configuration
`/opt/denodo/conf`	The configuration files of Denodo are stored in this folder
`/opt/denodo/lib-external`	Contains JDBC drivers not distributed with Denodo
`/opt/denodo/lib/data-catalog-extensions`	Contains Jar libraries used by Data Catalog
`/opt/denodo/lib/scheduler-extensions`	Contains Jar libraries used by Scheduler
`/opt/denodo/lib/solution-manager-extensions`	Contains Jar libraries used by Solution Manager (used in Solution Manager containers only)
`/opt/denodo/logs`	The logs are stored in this folder (check the section below for more information on this)
`/opt/denodo/metadata`	The metadata of our Denodo servers is saved in this folder
`/opt/denodo/resources/apache-tomcat`	The folder includes the configuration of the embedded Tomcat

`/opt/denodo/work/arn/data/index`	The indexes created with Scheduler Index are stored in this folder
`/opt/denodo/extensions/thirdparty/sap-jco`	Contains libraries needed by SAP BW and SAP BI data sources

Note that to mount the volumes we may need to change the ownership of the mounted folders. For more information see Mounting volumes to persist data.

2.2.1 Initialization of volumes in Kubernetes

The volumes in Kubernetes work in a different way than in Docker. When you create a volume in Kubernetes, the volume will be created as an empty folder. In many cases, the folder where we want to mount that volume has some content in the container image, so we expect this default content to be included in the volume. For instance, if you mount a volume in the `/opt/denodo/metadata` path of Denodo, you will notice that it will be empty instead of including the metadata structure distributed with the Denodo Platform.

With Docker new volumes are initialized with the contents of the folder where they are mounted, so, in order to achieve the same behavior with Kubernetes volumes, you must initialize the volumes explicitly when launching for the first time the pods that will use that volume, in a way that the volumes include the expected files that were distributed with the Denodo container image.

The below YAML shows a deployment of a Denodo container that is mounting the folder `/opt/denodo/metadata` in a volume. Notice that without the initContainers section of the deployment, the pod will not be able to start, since with an empty `metadata` folder the Denodo application won't work:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: denodo-deployment
spec:
  selector:
    matchLabels:
      app: denodo-app
  replicas: 1
  template:
    metadata:
      labels:
        app: denodo-app
    spec:
      hostname: denodo-hostname
      initContainers:
      - name: init-volume
        image: denodo-platform:8.0-latest
         command: ["/bin/sh", "-c", 'if [ -z "$(ls -A /tmp)" ]; then cp -R
/opt/denodo/metadata/* /tmp/; fi']
        volumeMounts:
          - name:  metadata-volume
```

```
          mountPath:  /tmp
    containers:
    - name: denodo-container
      image: denodo-platform:8.0-latest
      command: ["./denodo-container-start.sh"]
      args: ["--vqlserver"]
      ports:
      - name: denodo-port
        containerPort: 9999
      - name: web-container
        containerPort: 9090
      volumeMounts:
      - name: config-volume
        mountPath: /opt/denodo/conf/denodo.lic
        subPath: denodo.lic
      - name: metadata-volume
        mountPath: /opt/denodo/metadata
    volumes:
    - name: config-volume
      configMap:
        name: denodo-license
    - name: metadata-volume
```

Basically, to initialize the volume we are mounting it first with a different container on a different path, and copying all the contents from the folder we want to mount only if the volume is empty. That way, the next time the volume is mounted, it will be ready to be used in our main container including all the default configuration and files coming from the original image.

2.3 PERSISTING DENODO LOGS

If a containerized application unexpectedly ends you will probably need to check the logs, but if the logs will disappear with the container you won't probably be able to check them.

To solve these issues you need to choose a solution to manage your application logs, and Denodo allows you to do that in different ways. Some of the alternatives for managing logs in Denodo:

- Push the Denodo logs to your log aggregation system. For example, you can use Amazon CloudWatch to monitor Denodo
- Store your logs in an external system. For example, you could store Denodo logs in Amazon S3
- Use volumes to persist logs. In this scenario, you need to create a persistent volume for the `/opt/denodo/logs`, but please consider that if you have multiple containers logging to this volume you can have conflicts with the log files.
- Change the log configuration (`log4j2.xml` files) to output data to the standard output stream by using a `ConsoleAppender`. If you are redirecting multiple logs to the standard output you should modify the `PatternLayout` to include a

reference to the component that has produced that log entry. This is the default log mechanism supported by container engines and implies that you will need to use their tools to check the application logs.
For instance, in order to redirect the `vdp.log` file content to the standard output so `docker logs` can show you the lines, you can just replace the `Root` logger content in the Log4j configuration file `/opt/denodo/conf/vdp/log4j2.xml`:

```
...
<Root level="error">
    <AppenderRef ref="FILEOUT" />
</Root>
...
```

With:

```
...
<Root level="error">
    <AppenderRef ref="STDOUT" />
</Root>
...
```

- Use dedicated sidecar containers for log management. Although the denodo logging system is flexible, you may have specific needs that can be solved by using sidecar containers with a logging agent. You can read about this in the Kubernetes Logging Architecture document.

- Modify the Denodo startup script used inside the container to add a command for printing the content of the log files to the standard output, a command like the following one: `tail -n +1 -F ${DENODO_HOME}/logs/*/*.log &`
In this case, you should also update the `PatternLayout` in the log configuration files to include a reference to the component that has produced that log entry, but if you are going to modify the log configuration files it is easier to just select the option of using a dedicated `ConsoleAppender`.