Deploying Denodo in Kubernetes

Revision 20210615

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....2

2 DEPLOYING IN KUBERNETES....3
2.1 LICENSE CONFIGMAP....3
2.2 DENODO KUBERNETES SERVICE....3
2.3 TESTING....7
2.4 CLEAN UP....9
2.5 KUBERNETES AND CPUS....9

3 LICENSES MANAGED BY A DENODO SOLUTION MANAGER...10

4 TROUBLESHOOTING....11

5 REFERENCES....12

1 INTRODUCTION

The Denodo Platform can be containerized to be run in a container platform such as Docker. The usage of containers eases the adoption of modern architectures, for instance, microservices, which can be orchestrated with Kubernetes.

This document explains how to deploy Denodo containers using Kubernetes.

2 DEPLOYING IN KUBERNETES

Kubernetes is an orchestration system for containers, it allows the IT teams not only to manage the deployment of containerized applications but also to scale deployments by increasing the number of replicas for the deployment. Kubernetes also allows other actions, for instance, to update the current containers with its new version released.

This document will make use of the Kubernetes command-line tool (`kubectl`) that interacts with the Kubernetes cluster via the Kubernetes API. This article assumes that Kubernetes and Docker are already installed and working in your local environment and that Kubernetes is enabled on Docker, as this enables a single-node cluster when Docker is started. It is also recommended to follow the Denodo Platform Container QuickStart Guide as a prerequisite for this article, as it serves to check that the Denodo container is working successfully in the Docker installation. The Denodo users that are new to Kubernetes can also check the Kubernetes Basics tutorial to learn the basic concepts and usage of Kubernetes. The "`kubectl`" command-line tool that

communicates with Kubernetes to execute commands in the cluster.

2.1 LICENSE CONFIGMAP

The Denodo Platform requires a valid license in order to start, so in order to load this license we are going to create a configmap that will embed the file. Hence, the following statement creates the map with the contents of the license file that will be referenced later from the `denodo-service.yaml` file:

```
$ kubectl create configmap denodo-license --from-file=denodo.lic=<pathToLicenseFile>
```

2.2 DENODO KUBERNETES SERVICE

The Denodo Platform 8.0 introduced a new RMI implementation which eases the network configuration of the Virtual DataPort servers, requiring only one single port for establishing the connection with the Virtual DataPort Administration Tool and JDBC clients, but for JMX connections it still needs the two standard RMI Registry and RMI Factory ports. Due to these configuration differences between Denodo 8.0 and Denodo 7.0, we are providing one YAML file per version.

The following YAML configuration file defines a Kubernetes Service and Deployment that will deploy the Denodo Platform 8.0 container generated by Denodo in the Kubernetes cluster:

```
apiVersion: v1
kind: Service
metadata:
  name: denodo-service
spec:
  selector:
    app: denodo-app
  ports:
  - name: svc-denodo
    protocol: "TCP"
    port: 8999
    targetPort: denodo-port
  - name: svc-rmi-r
    protocol: "TCP"
    port: 8997
    targetPort: jdbc-rmi-rgstry
  - name: svc-rmi-f
    protocol: "TCP"
    port: 8995
    targetPort: jdbc-rmi-fctory
  - name: svc-odbc
    protocol: "TCP"
    port: 8996
    targetPort: odbc
  - name: svc-web
    protocol: "TCP"
    port: 8090
    targetPort: web-container
  type: LoadBalancer
```

```
---
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
      containers:
      - name: denodo-container
        image: denodo-platform:8.0-latest
        command: ["./denodo-container-start.sh"]
        args: ["--vqlserver"]
        env:
        - name: FACTORY_PORT
          value: "8995"
        ports:
        - name: denodo-port
          containerPort: 9999
        - name: jdbc-rmi-rgstry
          containerPort: 9997
        - name: jdbc-rmi-fctory
          containerPort: 8995
        - name: odbc
          containerPort: 9996
        - name: web-container
          containerPort: 9090
        volumeMounts:
        - name: config-volume
          mountPath: /opt/denodo/conf/denodo.lic
          subPath: denodo.lic
      volumes:
      - name: config-volume
        configMap:
          name: denodo-license
```

denodo-service.yaml for Denodo 8.0

The following YAML configuration file defines a similar Kubernetes Service and Deployment than the previous one but for a Denodo Platform 7.0 container:

```
apiVersion: v1
kind: Service
metadata:
  name: denodo-service
spec:
```

```
  selector:
    app: denodo-app
  ports:
  - name: svc-rmi-r
    protocol: "TCP"
    port: 8999
    targetPort: jdbc-rmi-rgstry
  - name: svc-rmi-f
    protocol: "TCP"
    port: 8997
    targetPort: jdbc-rmi-fctory
  - name: svc-odbc
    protocol: "TCP"
    port: 8996
    targetPort: odbc
  - name: svc-web
    protocol: "TCP"
    port: 8090
    targetPort: web-container
  type: LoadBalancer
---
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
      containers:
      - name: denodo-container
        image: denodo-platform:7.0-latest
        command: ["./denodo-container-start.sh"]
        args: ["--vqlserver"]
        env:
        - name: FACTORY_PORT
          value: "8997"
        ports:
        - name: jdbc-rmi-rgstry
          containerPort: 9999
        - name: jdbc-rmi-fctory
          containerPort: 8997
        - name: odbc
          containerPort: 9996
        - name: web-container
          containerPort: 9090
        volumeMounts:
```

```
        - name: config-volume
          mountPath: /opt/denodo/conf/denodo.lic
          subPath: denodo.lic
      volumes:
      - name: config-volume
        configMap:
          name: denodo-license
```

denodo-service.yaml for Denodo 7.0

If you are using your own custom Denodo containers, you need to change both the "`command:`" and "`agrs:`" lines in the previous YAML files to match the requirements from your containers.

To create both elements, service, and deployment, in the Kubernetes environment save the script to a file and name it to something like `denodo_service.yaml`. Then, execute the following Kubernetes command in a console:

```
> kubectl create -f denodo_service.yaml
```

```
C:\Denodo\Kubernetes>kubectl create -f denodo-service.yaml
service "denodo-service" created
deployment.apps "denodo-deployment" created

C:\Denodo\Kubernetes>kubectl get service
NAME             TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)                                                  AGE
denodo-service   LoadBalancer   10.109.211.35   localhost     8999:30651/TCP,8997:32632/TCP,8996:30442/TCP,8090:30349/TCP   13s
kubernetes       ClusterIP      10.96.0.1       <none>        443/TCP                                                  29d

C:\Denodo\Kubernetes>kubectl get deployment
NAME                DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
denodo-deployment   1         1         1            1           23s

C:\Denodo\Kubernetes>kubectl get pods
NAME                                 READY     STATUS    RESTARTS   AGE
denodo-deployment-68f744ff8-znw67    1/1       Running   0          28s

C:\Denodo\Kubernetes>
```

Execution of the denodo-service.yaml

Although this article does not try to explain how Kubernetes works or how YAML files are created, it is interesting to outline the following from the YAML file definition:

- The service for Denodo 8.0 exposes five ports: 8999, 8997, 8996, 8995 and 8090. These are the ports published in the service that are mapped to ports in the container. This mapping allows the use of the port 9090 in the container but publishes it as 8090 in the service. This is also done with other ports, however, the RMI factory port, in the example 8995, has to be the same in both the Pod and the Service because due to the internal workings of RMI connections this port cannot be mapped. If the RMI factory port has to be changed, it must be done at the Pod level (with appropriate container options).
- The configuration of the YAML file is assuming that a license is available in the configmap `denodo-license`. See section "License configmap" above for details on how to create one.
- All the Denodo pods will share the hostname *denodo-hostname*, as this is the selected solution to configure RMI appropriately with the standard Denodo container. The RMI configuration will be made automatically by the startup script that is included in the container using the hostname. Notice that the field

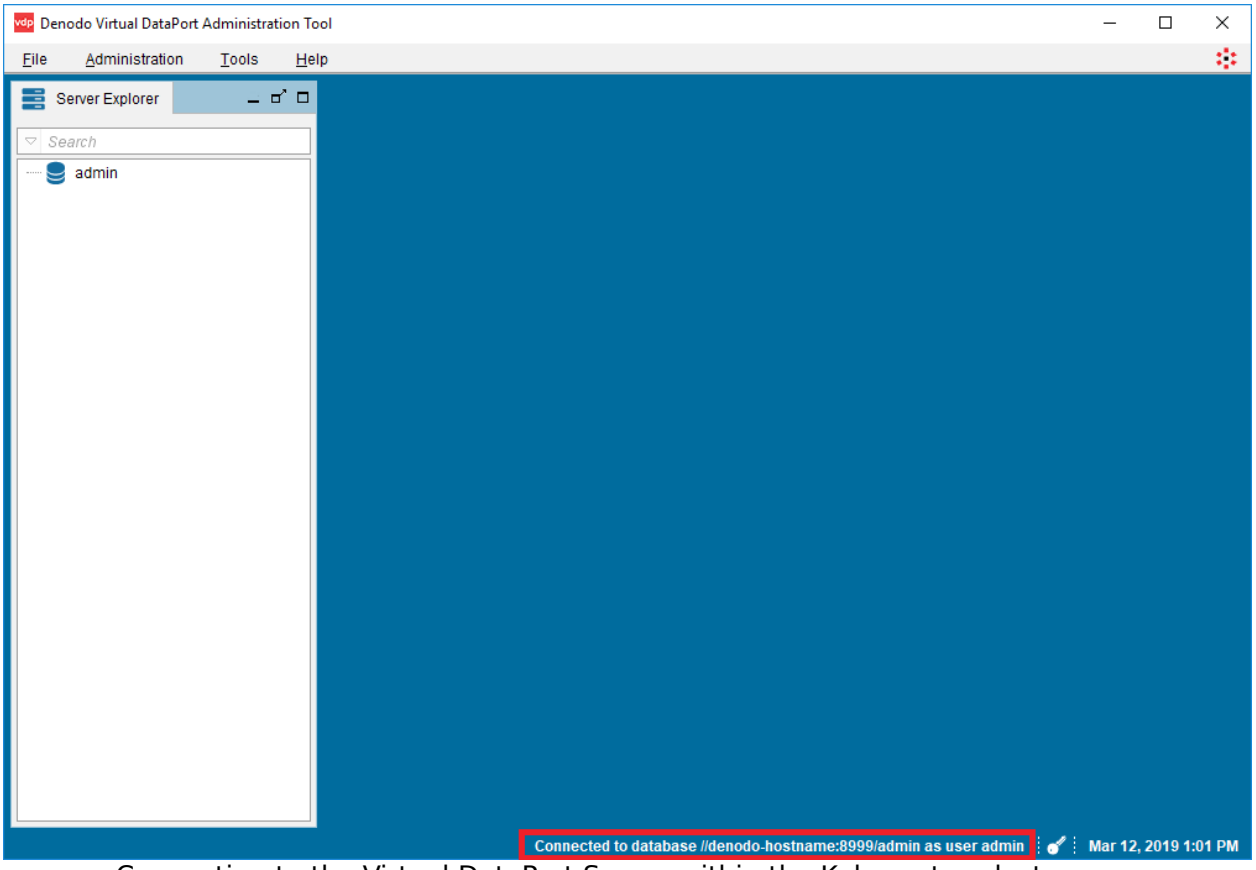
hostname cannot include dots, but If needed, Kubernetes also provides a subdomain field so you can assign a FQDN to the pod.

2.3 TESTING

After the Service deployment is completed, it will be possible to connect to the new Denodo instance from a Virtual DataPort client such as the Virtual DataPort Administration Tool with the following URL:

```
//denodo-hostname:8999/admin
```

In addition, notice that for using *denodo-hostname* as the connection host in your client computers, it will be necessary that the client computer is able to resolve that hostname to the IP address of the *denodo-service*, either by defining the *denodo-hostname* in the DNS server or in the *hosts* file of the client computer. Please, ensure that you have network connectivity from the client computer to the Kubernetes Service, by configuring the network routes appropriately.



Connecting to the Virtual DataPort Server within the Kubernetes cluster

The following command will provide all the information required regarding the Kubernetes Service that is deployed:

```
> kubectl describe service denodo-service
```

```
Name:                     denodo-service
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=denodo-app
Type:                     LoadBalancer
IP:                       10.106.74.98
LoadBalancer Ingress:     localhost
Port:                     svc-denodo  8999/TCP
TargetPort:               denodo-port/TCP
NodePort:                 svc-rmi  31560/TCP
Endpoints:                10.1.0.74:9999
Port:                     svc-rmi-r  8997/TCP
TargetPort:               jdbc-rmi-rgstry/TCP
NodePort:                 svc-rmi-r  31649/TCP
Endpoints:                10.1.0.74:9997
Port:                     svc-rmi-f  8995/TCP
TargetPort:               jdbc-rmi-fctory/TCP
NodePort:                 svc-rmi-f  30003/TCP
Endpoints:                10.1.0.74:8995
Port:                     svc-odbc  8996/TCP
TargetPort:               odbc/TCP
NodePort:                 svc-odbc  30715/TCP
Endpoints:                10.1.0.74:9996
Port:                     svc-web  8090/TCP
TargetPort:               web-container/TCP
NodePort:                 svc-web  30118/TCP
Endpoints:                10.1.0.74:9090
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```

kubectl describe denodo-service output

2.4 CLEAN UP

Finally, in case that it is needed to clean the environment, the following commands can be executed to delete the configmap for the license and the Denodo service deployed in Kubernetes:

```
> kubectl delete -f denodo-service.yaml
> kubectl delete configmap solution-manager-license
```

2.5 KUBERNETES AND CPUS

By default, the pods running in Kubernetes are able to view as available all the cores from the node where the pod is running. If you are using a Denodo license with core restrictions, you might find that a server cannot start because the cores available in the pod exceeds the license restriction.

You can check the number of cores in your nodes with the following command:

```
kubectl get nodes -o custom-columns=NODE:.metadata.name,CPU:.status.capacity.cpu
```

In the case that you can not adjust the capacity of the nodes in your cluster, the KB Article How to limit the number of CPUs in Kubernetes explains how to reconfigure your nodes to allow you to limit the number of cores por a pod in Kubernetes, so you can start Denodo in nodes that have more cores than those allowed by the license.

3 LICENSES MANAGED BY A DENODO SOLUTION MANAGER

The Denodo Solution Manager helps large organizations to manage the Denodo licensing in an easy way, for instance, it can serve licenses to the servers running on Kubernetes. However, due to the containers and Kubernetes architecture, we will not use it for deploying new revisions directly on running containers, instead, we will follow the Kubernetes best practices by creating new images and updating the deployments accordingly. If desired, the Solution Manager features around revisions can still be used to generate these images from non containerized environments.

In order to get the license from the Solution Manager, we need to apply some changes to the pod configuration. The script provided above to create the Kubernetes deployment makes use of a configmap with a standalone license. However, it is possible to apply some changes to it in order to retrieve the license information from a License Server instead of pointing to a license file. To do this, just make the following replacements in the script file:

Replace the last part of the file:

```
        volumeMounts:
        - name: config-volume
          mountPath: /opt/denodo/conf/denodo.lic
          subPath: denodo.lic
      volumes:
      - name: config-volume
        configMap:
          name: denodo-license
```

with

```
        volumeMounts:
        - name: config-volume
          mountPath: /opt/denodo/conf/SolutionManager.properties
          subPath: SolutionManager.properties
      volumes:
      - name: config-volume
        configMap:
          name: solution-manager
```

In the replacement, you will notice that instead of mounting the configmap `denodo-license` we are mounting the configmap `solution-manager`. Notice that this configmap has to include the Solution Manager configuration file `SolutionManager.properties`, and it can be created with the following command:

```
$ kubectl create configmap solution-manager --from-file=SolutionManager.properties=<pathToConfigurationFile>
```

The Solution Manager configuration file `SolutionManager.properties` should point to the Solution Manager server, with a configuration like the one shown here:

```
# License Manager Configuration
com.denodo.license.host=solutionmanager-hostname
com.denodo.license.port=10091
```

The hostname *solutionmanager-hostname* points to the Solution Manager server and it has to be resolved by the pods. Also, in the Solution Manager side, it will be required to create the Kubernetes environment with the Virtual DataPort servers setting the Host parameter as *denodo-hostname*.

In case you are wondering if the Denodo Solution Manager can be deployed in Kubernetes, we also have a specific KB article showing how to deploy the Denodo Solution Manager in Kubernetes.

4 TROUBLESHOOTING

Sometimes we can face issues when launching the pods, for instance, if the Virtual DataPort server cannot start because there is a misconfiguration with the license.

The following list includes some common errors that are faced when working with Denodo and Kubernetes:

- `CrashLoopBackOff`: this error appears when there is a problem with the configuration of the license file and the Virtual DataPort server cannot start. In order to check what is happening, the simplest solution is to use the parameter `terminationMessagePath` explained below to get more information about the issue.
- `ErrImagePull`: in this case Kubernetes cannot pull the image due to network issues or an incorrect configuration of the registry credentials. Check the image pull policy defined in the YAML is the correct one.
- `ImagePullBackOff`: another problem related with the pull operation of the images, we can see it when Kubernetes cannot find the image in the local or public registry. Many times this problem appears because the Denodo Docker image has not been loaded locally with a `docker load` so Kubernetes cannot run the container.
- `RunContainerError`: usually obtained when deploying a YAML file with configuration errors, executing a `kubectl describe pod` will help diagnosing the issue.

In order to diagnose an issue when starting a Denodo container without doing any changes to the container image, nor having to open a shell inside the container to view the logs, it is very useful to use the `terminationMessagePath` field in order to customize the termination message of the pod. The field can be added within the container information of the YAML, so the content of a Denodo log file in the container is taken as hint for the pod termination information:

```
...
       containers:
      - name: denodo-container
        image: denodo-platform:8.0-latest
        terminationMessagePath: "/opt/denodo/logs/vdp/vdp.log"
...
```

Thus, we can set `terminationMessagePath` to the Virtual DataPort log file `vdp.log`, so if the pod is not starting due to a license problem that log will tell us so.

Please note that this option truncates the content of the indicated file to generate the termination message, and although the truncated size is enough to diagnose pod startup issues it will provide misleading information in other situations, so this should only be used while diagnosing startup problems.

With this configuration, if for any reason the pod fails when starting, in order to see the message from the log in Kubernetes, we can use `kubectl describe pod` to review the termination message of the pod:

```
> kubectl describe pod <denodo-deployment-pod>
```

5 REFERENCES

Denodo Platform Container QuickStart Guide
Kubernetes Documentation
Learn Kubernetes Basics
Kubernetes on Docker