Deploying Denodo in the Azure Kubernetes Service (AKS)

Revision 20200814

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

The Azure Kubernetes Service (AKS) is a service provided by Azure that supports deploying Kubernetes clusters in Azure. AKS has become the default technology when working with Azure + Kubernetes due to its ease of use and its integration features with other Azure services such as the Azure Active Directory.

In this article we will show how to create a Kubernetes cluster in AKS using the Azure Command Line Interface, and how to deploy a Denodo Kubernetes application in it.

1 AZURE CLI

The Azure CLI is a command-line tool that can be used for administering the Azure resources and we will use it extensively in the article, we will assume that it is installed and up to date as a prerequisite for executing the commands presented below.

Although it is also possible to manage the AKS resources from the Azure portal, in this article we have chosen to use the Azure CLI because usually, the commands are self-explanatory and also because it is possible to use these commands and test them easily in any custom Azure deployment.

2 AZURE KUBERNETES SERVICE

The Azure Kubernetes Service eases the management, configuration and daily work of containerized environments in Azure. In other Knowledge Base articles, we explained how to deploy a Denodo container in a local Kubernetes environment and now, we will see how to perform this deployment into a cluster that is hosted in Microsoft Azure, making use of the Azure Kubernetes Service.

Note that all the below statements are Azure CLI commands, they can be copied into any environment to create a Denodo cluster in Azure.

The cluster creation is organized in three main parts:

- Creation of the container registry: the Kubernetes cluster will use an Azure Container Registry to obtain the container images.
- Creation of the Kubernetes cluster: the cluster will accept the deployment of the Denodo image.
- Deployment of Virtual DataPort in the cluster: the last step shows the deployment of the Denodo image, explaining the implications of doing this in Azure.

2.1 LIST OF PREREQUISITES

This article assumes that there is an environment already configured to deploy a Docker image of Denodo in AKS, in other cases, there might be needed to install or configure something else before continuing with the next section. In summary:

- The Azure CLI and a valid Azure account with enough privileges to build all the Azure elements involved in the deployment of a Denodo image.
- A Docker image of Denodo in the local registry.
- Additionally, it will be easier to configure the deployment file if a local Kubernetes environment is created first, with a tested `denodo-service.yaml`.

2.2 CREATE THE CONTAINER REGISTRY

To start with the Azure configuration, open a new console and execute the below Azure commands that will perform the following actions:

Log in to Azure interactively

```
$ az login
```

Create a new resources group to use with the Azure resources that will be created in this guide. Provide a custom name and a valid location, for instance *francecentral*. It is possible to obtain the list of valid locations with the “list-locations” command.

```
$ az account list-locations
$ az group create --name <denodoResourceGroup> --location <location>
```

Create a new Azure Container Registry by providing a custom name and the name of the Resources Group created in the previous step.

```
$ az acr create --resource-group <denodoResourceGroup> --name
<denodoAzureContainerRegistry> --sku Basic
```

Then, log in to the newly created Azure Container Registry, get the login server address of the registry and tag the Denodo local image with the login server obtained.

```
$ az acr login --name <denodoAzureContainerRegistry>
$ az acr list --resource-group <denodoResourceGroup> --query "[].
{acrLoginServer:loginServer}" --output tsv
$ docker tag denodo-platform <acrLoginServer>/denodo-platform:v1
```

Finally, push the image tagged to the Azure Container Registry by providing the ACR login server. List the registry to verify that the image is available.

```
$ docker push <acrLoginServer>/denodo-platform:v1
$ az acr repository list --name <denodoAzureContainerRegistry> --output
table
```

2.3 CREATE THE KUBERNETES CLUSTER

The AKS cluster privileges can be managed with role-based access controls, which is quite helpful since it allows to define the access to the resources based on roles assigned to users.

Before creating the cluster we will prepare a new service principal that we will set as a parameter in the cluster creation statement. Notice that to create the service principal we can provide a custom name, and Azure will return the identifier for this new principal, and it is the identifier that we will use later in the other Azure statements to refer to this service principal.

The next step is to retrieve the identifier of the Azure container registry created previously, in order to assign the role `acrpull` to the service principal over the container registry, so the cluster can pull images from the registry.

Finally, we can create the Kubernetes cluster by providing: the name for the cluster, the resource group and the service principal identifier and secret.

```
$ az ad sp create-for-rbac --skip-assignment --name <denodoServicePrincipal>
$ az acr show --resource-group <denodoResourceGroup> --name
<denodoAzureContainerRegistry> --query "id" --output tsv
$ az role assignment create --assignee <servicePrincipalId> --scope <acrId>
--role acrpull
$ az aks create \
    --resource-group <denodoResourceGroup> \
    --name <denodoAKSCluster> \
```

```
    --node-count 1 \
    --service-principal <servicePrincipalId> \
    --client-secret <clientSecret> \
    --generate-ssh-keys
```

The cluster is now running in Azure AKS and we can connect to it with the default Kubernetes CLI `kubectl`.

In order to do that we have to install the Kubernetes CLI and configure the connection to the remote cluster in Azure. The latter is done with the `az aks get-credentials` command, where you need to provide the Azure cluster name and the name of its resource group.

To check if the configuration is successful, connect to the cluster and get the nodes information with `kubectl`.

```
$ az aks install-cli
$ az aks get-credentials --resource-group <denodoResourceGroup> --name
<denodoAKSCluster>
$ kubectl get nodes
```

2.4 DEPLOYING DENODO VIRTUAL DATAPORT

The Denodo Platform requires a valid license in order to start, which can be obtained from a Denodo License Manager server or in some scenarios (like evaluation or Denodo Express) as a standalone license file. In order to avoid static references, we can use a Kubernetes config map that will embed a Solution Manager configuration file pointing to the License server or a valid license file.

2.4.1 Deploy Virtual DataPort in a Solution Manager environment

If we have a Denodo License Manager server running and accessible by the Kubernetes cluster then it can be configured as part of the Solution Manager configuration and we can use it to retrieve a valid license.

First, we need to create the config map containing the Solution Manager configuration that will be referenced later from the `denodo-service.yaml` file:

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

After this, we can deploy the Virtual DataPort server in the AKS cluster. Taking as starting point the `denodo-service.yaml` from the Knowledge Base article Deploying Denodo in Kubernetes, we will have to update two sections of the base file in order to accomplish a successful deployment.

The first modification will consist in setting up correctly the image name, so you need to replace the following line in the `denodo-service.yaml`:

```
image: denodo-platform:8.0-latest
```

with

```
image: <acrLoginServer>/denodo-platform:v1
```

Notice that the value of the placeholder `<acrLoginServer>` was already obtained in a previous step and it refers to the login server address of your registry.

In addition, it is required to modify the YAML file in order to copy the solution manager configuration file from the config map to the right location. The reason is that in the original YAML file, the configuration file was mounted directly from the local file system, but in this case we will load the file from the config map `solution-manager` that we have just created with the previous kubectl statement.

Hence, after applying the mentioned updates, the Denodo 8.0 version of the file would look like this:

```
apiVersion: v1
kind: Service
metadata:
  name: denodo-service
spec:
  selector:
    app: denodo-app
  ports:
  - name: svc-rmi
    protocol: "TCP"
    port: 8999
    targetPort: jdbc-rmi
  - 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: <acrLoginServer>/denodo-platform:v1
        command: ["./denodo-container-start.sh"]
        args: ["--vqlserver"]
        env:
        - name: FACTORY_PORT
          value: "8995"
        ports:
        - name: jdbc-rmi
          containerPort: 9999
        - name: jdbc-rmi-rgstry
          containerPort: 9997
        - name: jdbc-rmi-fctory
          containerPort: 8995
        - name: odbc
          containerPort: 9996
        - name: web-container
          containerPort: 9090
        lifecycle:
          postStart:
            exec:
                                              command:   ["/bin/sh",   "-c",   "cp
/opt/denodo/sm/SolutionManager.properties
/opt/denodo/conf/SolutionManager.properties"]
        volumeMounts:
        - name: config-volume
          mountPath: /opt/denodo/sm
          readOnly: true
      volumes:
      - name: config-volume
        configMap:
          name: solution-manager
          items:
          - key: SolutionManager.properties
            path: SolutionManager.properties
```

denodo-service.yaml for Denodo Platform 8.0

On the other hand, this would be the *denodo-service.yaml* version for Denodo Platform 7.0:

```
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: <acrLoginServer>/denodo-platform:v1
        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
        lifecycle:
          postStart:
            exec:
                                          command:   ["/bin/sh",   "-c",   "cp
/opt/denodo/sm/SolutionManager.properties
/opt/denodo/conf/SolutionManager.properties"]
        volumeMounts:
        - name: config-volume
          mountPath: /opt/denodo/sm
          readOnly: true
      volumes:
      - name: config-volume
        configMap:
          name: solution-manager
          items:
          - key: SolutionManager.properties
            path: SolutionManager.properties
```

denodo-service.yaml for Denodo Platform 7.0

Then, take the appropriate version of the *denodo-service.yaml* and deploy the app and service in Kubernetes with the following command:

```
$ kubectl apply -f denodo-service.yaml
```

After all these steps a Virtual DataPort server will be running in the AKS cluster that we created. Azure will provide a public IP address for the Kubernetes service that can be used for connecting to Virtual DataPort. In order to know the address, we can use the following command to obtain the public IP address for the Kubernetes service:

```
$ kubectl get service denodo-service
```

```
C:\>kubectl get service denodo-service
NAME             TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)                                                                  AGE
denodo-service   LoadBalancer   10.0.97.174   [redacted]      8999:30679/TCP,8997:31758/TCP,8995:32375/TCP,8996:32323/TCP,8090:30890/TCP   3m16s
```

Output from the execution of the get service command

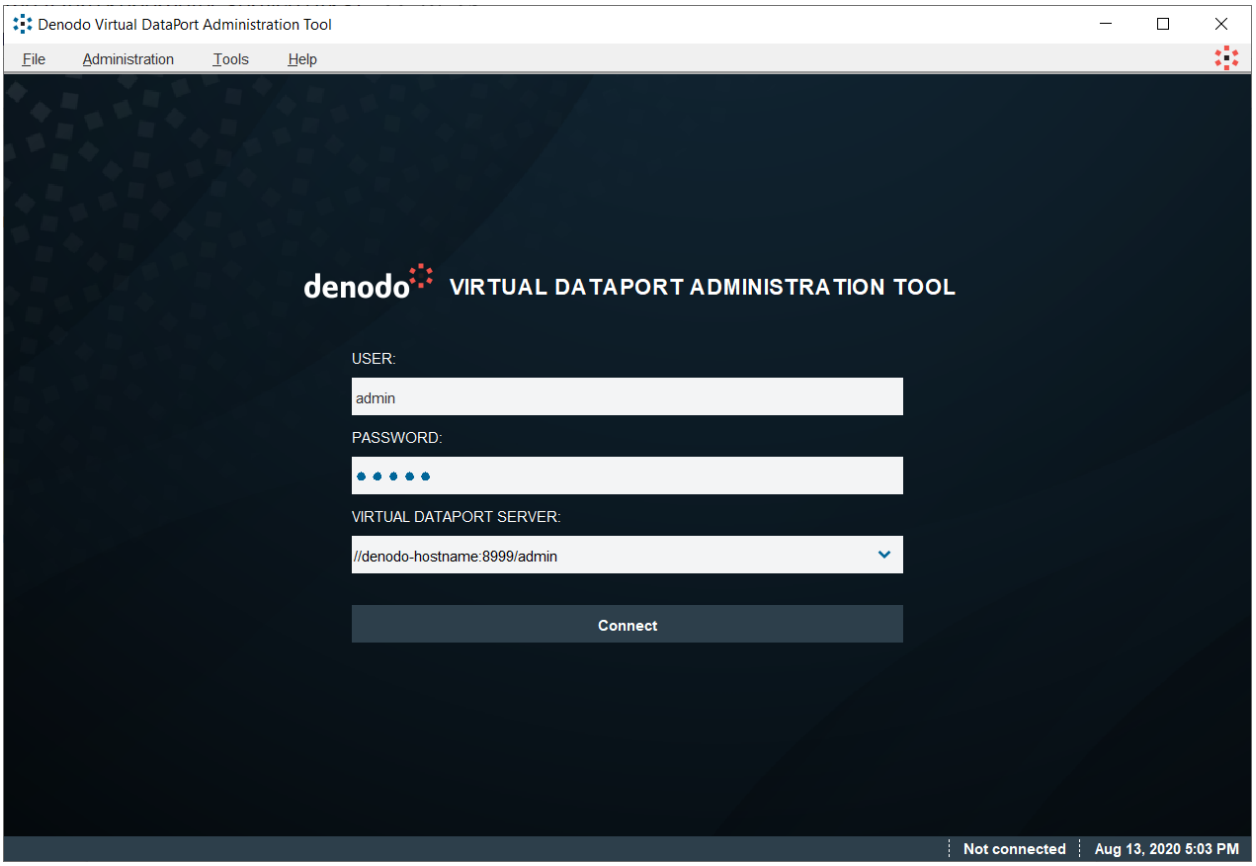
Also, notice that in the YAML configuration file, we are specifying the hostname for the Virtual DataPort server as `denodo-hostname`, so in order to connect to the server from a VDP client, we will need to ensure that the VDP client is able to resolve this hostname to the public IP address of the Kubernetes service, and we can do that by adding an entry in our local hosts configuration file to map that hostname with the IP address:

```
# Denodo Kubernetes service
20.30.40.50 denodo-hostname
```

The new entry in the Hosts file

Now we can open a new Virtual DataPort Administration Tool and connect to the server using the following Denodo Server URI:

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



2.4.2 Deploy Virtual DataPort with a standalone license

To use a standalone license file, we can use the following statement to create a 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>
```

Then, we need to modify the YAML file to load the license file in the Denodo installation folder `<DENODO_HOME>/conf` as it is done with the solution manager configuration. For this, replace the following part of the previous file `denodo-service.yaml`:

```
...
lifecycle:
```

```
          postStart:
            exec:
                                              command:   ["/bin/sh",   "-c",   "cp
/opt/denodo/sm/SolutionManager.properties
/opt/denodo/conf/SolutionManager.properties"]
        volumeMounts:
        - name: config-volume
          mountPath: /opt/denodo/sm
          readOnly: true
      volumes:
      - name: config-volume
        configMap:
          name: solution-manager
          items:
          - key: SolutionManager.properties
            path: SolutionManager.properties
```

with:

```
        ...
        lifecycle:
          postStart:
            exec:
                                              command:   ["/bin/sh",   "-c",   "cp
/opt/denodo/conf/license/denodo.lic /opt/denodo/conf/denodo.lic"]
        volumeMounts:
        - name: config-volume
          mountPath: /opt/denodo/conf/license
          readOnly: true
      volumes:
      - name: config-volume
        configMap:
          name: denodo-license
          items:
          - key: denodo.lic
            path: denodo.lic
```

3 REFERENCES

Azure CLI
Azure Kubernetes Service (AKS)
Deploying Denodo in Kubernetes
How to create your own Docker images for Denodo Platform Containers