Deploying Denodo in Google Kubernetes Engine (GKE)

Revision 20220816

CONTENTS

The Google Kubernetes Engine (GKE) is a service provided by Google that supports deploying Kubernetes clusters in Google Cloud. GKE has become the default technology when working with Google + Kubernetes due to its ease of use and its integration features with other Google Cloud services.

It is worth mentioning that Kubernetes was actually designed by Google, so although other cloud vendors offer alternative services to Kubernetes in their catalog, Google Cloud focuses on Kubernetes as their container orchestration platform of reference.

In this article, we will show how to create a Kubernetes cluster in GKE using the gcloud command-line tool, and how to deploy a Denodo Kubernetes application in it.

1 THE GCLOUD COMMAND-LINE TOOL

The Google Cloud Command Line Interface (gcloud CLI) is a command-line tool that can be used for administering the Google Cloud 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 GKE resources from the Google Cloud console, in this article we have chosen to use the gcloud 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 Google Cloud deployment.

2 GOOGLE KUBERNETES ENGINE

The Google Kubernetes Engine eases the management, configuration, and daily work of containerized environments in Google Cloud. 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 Google Cloud, making use of the Google Cloud Engine.

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

The cluster creation is organized into three main parts:

- Creation of the container registry: the Kubernetes cluster will use a Google 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 Google Cloud.

2.1 LIST OF PREREQUISITES

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

- The gcloud CLI and a valid Google Cloud account with enough privileges to build all the Google Cloud 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 Google Cloud configuration, open a new console and execute the below Google Cloud commands that will perform the following actions:

Set up your default credentials, zone and project configuration with the command `gcloud init` from the Google SDK:

```
$ gcloud init
```

Enable the Container Registry API so you can push images to the Container Registry.

```
$ gcloud services enable containerregistry.googleapis.com
```

Configure the credentials of gcloud in your Docker environment so you can push images to the container registry.

```
$ gcloud auth configure-docker
```

Tag the Denodo image and push it. The first image pushed will create the registry in the project, and the corresponding Cloud Storage bucket behind the scenes. To compose the registry URL, you need to replace <PROJECT_ID> with the project ID.

```
$ docker tag denodo-platform gcr.io/<PROJECT_ID>/denodo-platform
$ docker push gcr.io/<PROJECT_ID>/denodo-platform
```

You can list the registry to verify that the image is available.

```
$ gcloud container images list
$ gcloud container images list-tags gcr.io/<PROJECT_ID>/denodo-platform
```

2.3 CREATE THE KUBERNETES CLUSTER

By default, the service account configured on the workers of the GKE cluster can pull images from the Container Registry if it belongs to the same project, so in order to simplify the privileges configuration, we will create all the resources under the same Google Cloud project. In case that you need to use a different project, you can check the Google Container Registry documentation to check what privileges you need to configure.

Create the Kubernetes cluster by providing the name for the cluster and the number of nodes for each of the cluster's zones.

```
$ gcloud container clusters create <CLUSTER_NAME> --num-nodes=1
```

The cluster is now running in Google Cloud GKE 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 Google Cloud. The latter is done with the `gcloud container clusters get-credentials` command, where you need to provide the Google Cloud cluster name.

To check if the configuration is successful, connect to the cluster and get the node's information with kubectl.

```
$ gcloud container clusters get-credentials <CLUSTER_NAME>
$ 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 GKE 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 of 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: gcr.io/<PROJECT_ID>/denodo-platform
```

Notice that we have already used previously the value of the placeholder `<PROJECT_ID>` and it refers to the project ID.

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-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: gcr.io/<PROJECT_ID>/denodo-platform
        command: ["/opt/denodo/tools/container/entrypoint.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
        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

NOTE: This YAML file only applies to the Denodo Containers released with Denodo 8.0 Update 20220815 and later.

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: gcr.io/<PROJECT_ID>/denodo-platform
        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
```

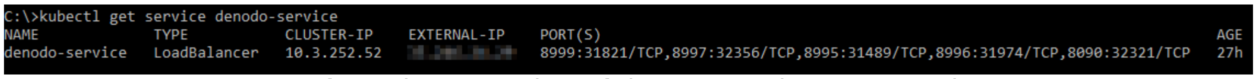
denodo-service.yaml for Denodo Platform 7.0

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 GKE cluster that we created. Google Cloud 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   10.3.252.52    [redacted]    8999:31821/TCP,8997:32356/TCP,8995:31489/TCP,8996:31974/TCP,8090:32321/TCP   27h
```

Output from the execution of the get service command

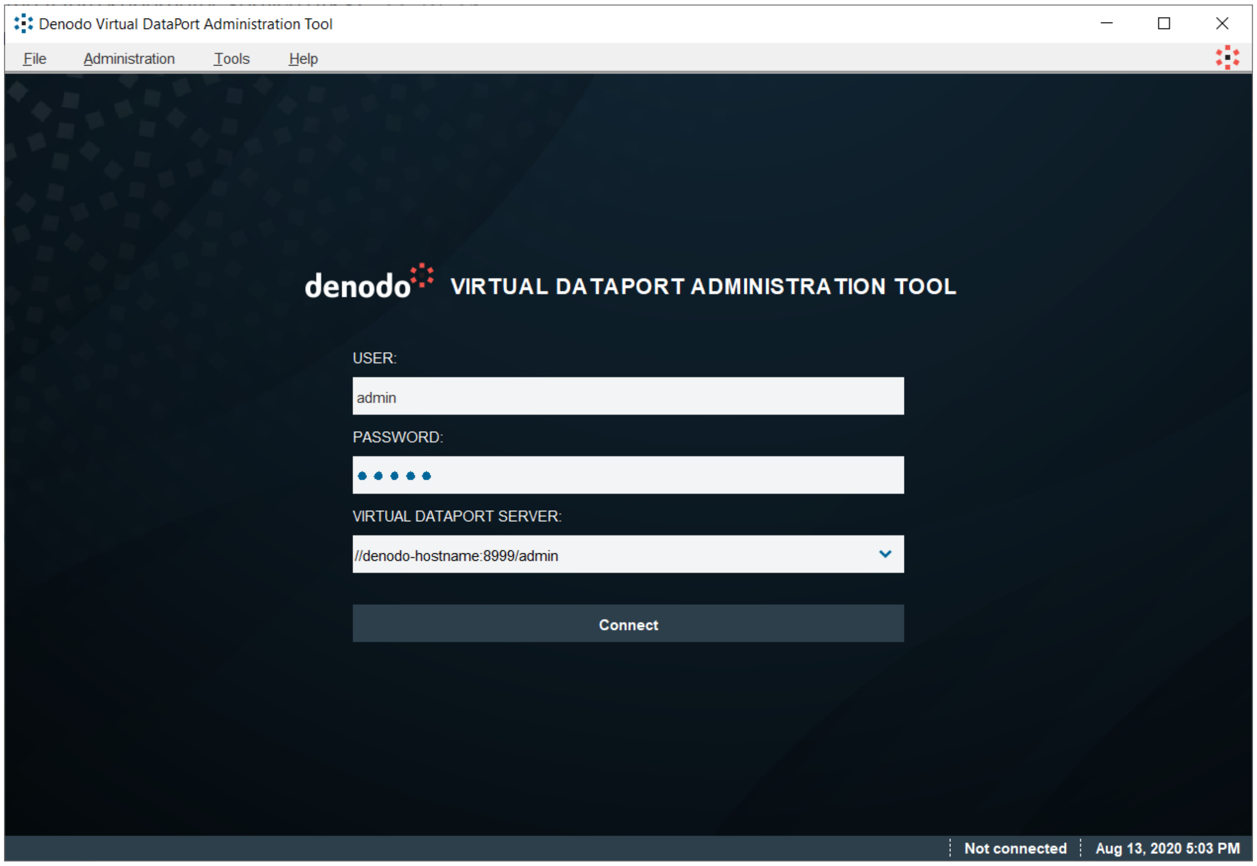
Also, notice that in the YAML configuration file, we are specifying the hostname for the Virtual DataPort server as `denodo-hostname`, so in order to connect to the server from a VDP client, we will need to ensure that the VDP client is able to resolve this hostname to the public IP address of the Kubernetes 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
```


Denodo Virtual DataPort Administration Tool
File
Administration
Tools
Help
denodo VIRTUAL DATAPORT ADMINISTRATION TOOL
USER:
admin
PASSWORD:
VIRTUAL DATAPORT SERVER:
//denodo-hostname:8999/admin
Connect
Not connected
Aug 13, 2020 5:03 PM

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

gcloud CLI
Google Kubernetes Engine (GKE)
Google Cloud Container Registry