Configuring Autoscaling of Denodo in Kubernetes

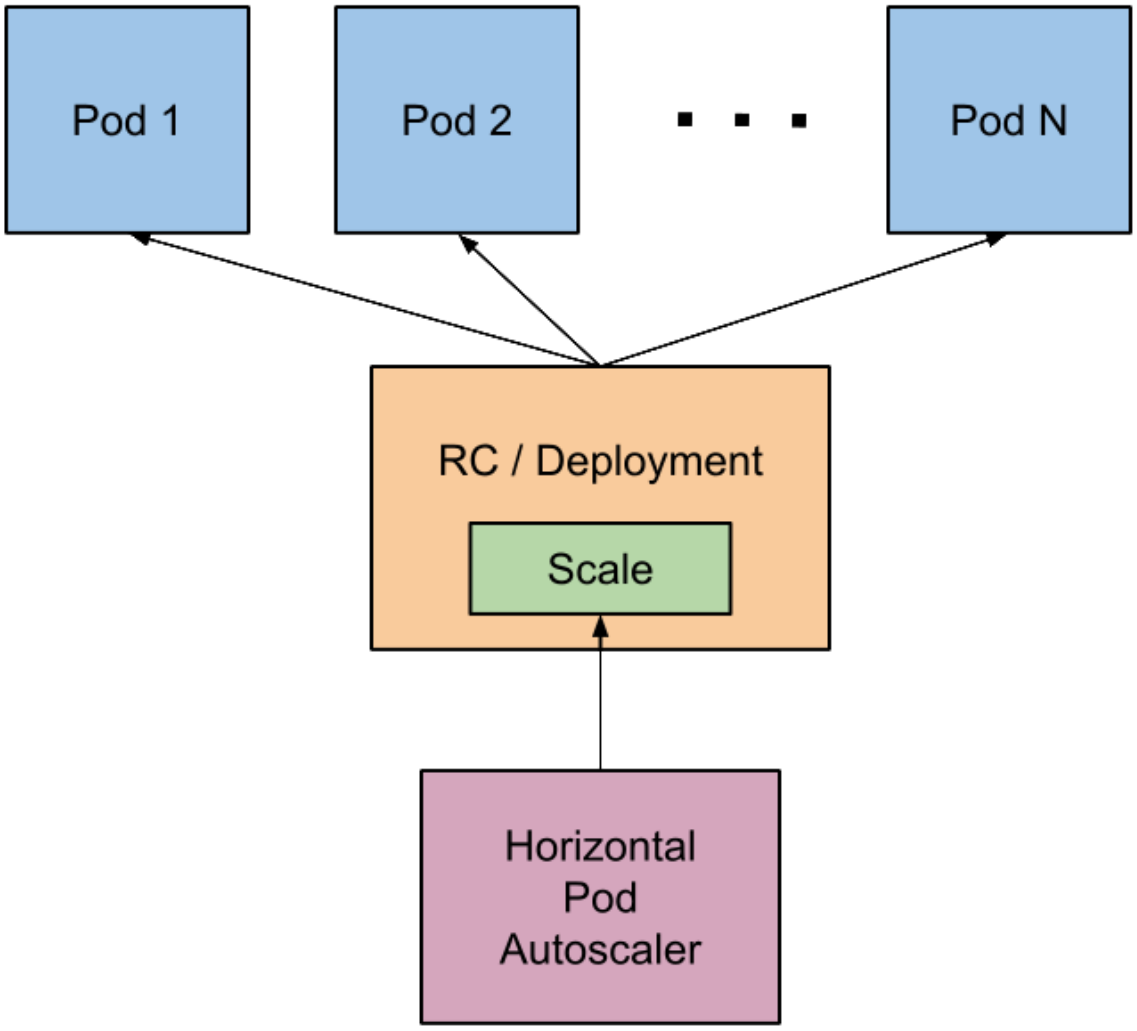
Revision 20200812

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

Kubernetes, the open-source container-orchestration system, supports the automation of autoscaling deployments. This means that a given deployment can increase or decrease the number of its pods to adapt the cluster resources to the business user's demand. For instance, you could define a default deployment of 2 replicas that could grow up to 4 pods if the resource usage is high enough.



The Horizontal Pod Autoscaler scales the pods based on metrics

Kubernetes encapsulates the autoscaling configuration within the autoscale object, which holds the scaling information such as the number of minimum and maximum pods for the cluster. So, if we already have a Denodo deployment running with name denodo-deployment, creating an autoscaler for the deployment is as easy as executing:

```
> kubectl autoscale deployment denodo-deployment --min=2 --max=10
```

The previous command enables the deployment to autoscale automatically between 2 and 10 pods. To decide how many replicas are needed in the cluster, the autoscaler will use a default algorithm that is based on CPU metrics from the pods.

However, in order to make the statement work, we first have to create a deployment that can be autoscaled, this is explained in detail in the following sections of the article.

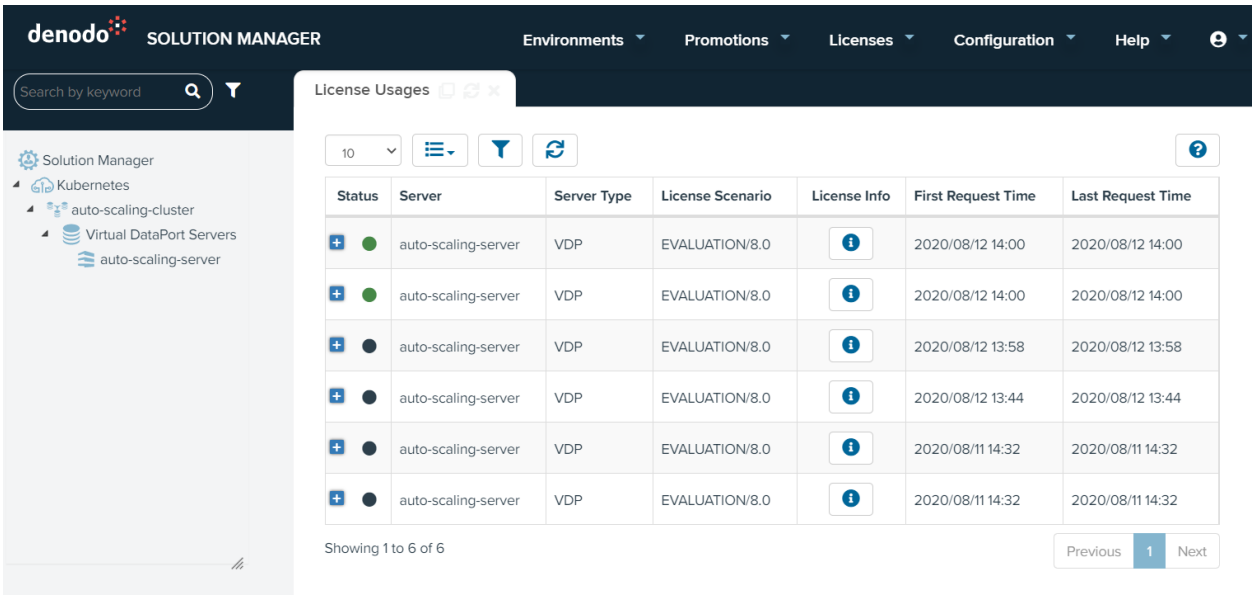
2 SUPPORTING THE AUTOSCALING PROCESS

The Denodo servers require valid licenses in order to start. In a real environment, the first thing a Denodo server does is connecting to a License Manager Server to fetch a license. For this to work, the server has to be previously declared in the Solution Manager, and so the License Manager will recognize the server that is connecting and serve a license to it.

In an autoscaling environment, we can take two different approaches to satisfy this requirement. On the one hand the Solution Manager can be prepared beforehand by declaring the needed server or, on the other hand, the server configuration can be declared dynamically at deployment time.

The example that we present assumes that the following environment, cluster and server will be created following some convention names:

- Environment: `auto-scaling-environment`
- Cluster: `auto-scaling-cluster`
- Server: `auto-scaling-server`



The Solution Manager showing 2 pods running in the cluster

However, for demonstration purposes, and in order to clarify the procedure in a more complex scenario, before deploying the cluster we check if the server is created in the Solution Manager, and in the case that it does not exist, we will create it. These actions have to take place before the Virtual DataPort server starts, so we will be using in our case the *initContainers* feature of Kubernetes.

The Solution Manager server can be created either by providing a hostname or an IP address. Hence, although using hostnames is usually a better option, we can choose what fits better our requirements. In the solution presented by this article, the server is created with the hostname *denodo-hostname*, so with just one single server and given that all the pods are launched with the same hostname, the License Manager will serve the licenses to several pods/instances as long as the license restrictions and capacity are satisfied. It is important to take into account that license restrictions may involve for instance, number of servers or number of cores, so while the total number of instances/cores are below the maximum number allowed in the license, Kubernetes will be able to add more pods to the cluster.

When the load of the cluster decreases, the auto scaling component can drain the cluster to reduce the pods number. The pods should release the license at this stage so the license is not blocked by a pod that is terminated and next pods can reuse that license that has been freed. With the *PreStop* hook of Kubernetes it is very easy to perform some action before the pod is stopped, for instance, we can invoke a script to release the license of the pod.

3 DENODO LICENSE MANAGER

The Denodo License Manager simplifies the management of licenses and it even supports complex environments with autoscaling clusters. For this, the Solution Manager 7.0 starting with the update 20190903 provides a REST API that allows updating the metadata by adding and removing servers, clusters, and environments.

As we have seen in the Denodo Knowledge Base article Deploying Denodo in Kubernetes, Denodo can be deployed in Kubernetes. However, in order to build autoscaling environments, we need to bear in mind that a Denodo server has to be registered in the Solution Manager to fetch a valid license. Without doing this, it will not be able to start.

Notice that the licenses distributed by Denodo may include restrictions regarding the maximum number of servers and/or total cores that can be launched at the same time. The License Manager will serve the licenses normally while the restrictions are satisfied. For instance, in an environment where we are running instances of 4 cores and where the license holds a restriction of maximum cores set to 12, it will be possible to run up to 3 servers of 4 cores.

With the Solution Manager REST API, we can ensure that a server is registered dynamically at pod launch time and that the license is released when the pod is therefore destroyed. Kubernetes provides several mechanisms that permit the pods to execute actions in different stages of the pod lifecycle, we will make use of them to register the server in the Solution Manager when the first pod is launched, and release the license from the server when each pod is removed.

4 KUBERNETES INIT CONTAINERS

Many times, the containers deployed in Kubernetes include no configuration and need to be set up before they run. The init containers are specialized containers that run before the app containers to configure the app images.

In our case, we will add an initContainers section to the Kubernetes YAML file to execute a script for registering the server in the Solution Manager in the case that it has not been registered previously. Hence, the Solution Manager will be ready to serve the license appropriately before the Virtual DataPort server is started.

The snippet will look like this:

```
...
initContainers:
  - name: register-server
    image: everpeace/curl-jq
    command: [sh, -c, "opt/denodo/kubernetes/register-server.sh"]
    envFrom:
    - configMapRef:
        name: server-config
    volumeMounts:
    - name: register-server
      mountPath: /opt/denodo/kubernetes/register-server.sh
...
```

In the code can be seen that the script is launched in the container *everpeace/curl-jq*, this is a lightweight container that has installed the tools *curl* to send Post requests and *jq* to manage JSON responses. The shell script *register-server.sh* adds the server to the License Manager (if the server does not exist yet) by invoking the License Manager REST API via curl. The script itself can be found at the end of the article.

5 KUBERNETES CONTAINER LIFECYCLE HOOKS

In addition to the init containers, Kubernetes also provides two hooks to run code at some concrete stage of the container lifecycle. The two hooks available are *PostStart* and *PreStop*. The former is executed just after creating the container but we cannot use it for registering the server since we could not ensure that the server would be registered before starting the server. The latter, *PreStop*, is invoked just before the container is stopped. We will use it for releasing the license of the server in the License Manager before stopping the server.

```
...
lifecycle:
  preStop:
    exec:
      command: [sh, -c, "kubernetes/release-license.sh"]
...
```

6 AUTOSCALING SAMPLE

The below scripts are a sample for building an autoscaling scenario for Denodo with Kubernetes. They assume that there is an environment and a cluster created appropriately. Also, the scripts expect the server to be contained in the cluster, but in case that it does not exist, the script will create the server before running the pod.

The `register-server.sh` invokes the Solution Manager API to create the server in the Solution Manager and `release-license.sh` invokes the API to release the license usage from the server.

The `denodo-service.yaml` creates all the Kubernetes objects, from the `ConfigMap` that embeds the `register-server.sh` and `release-license.sh` configuration to the denodo-deployment and denodo-service that create the cluster. It also includes the invocations to `register-server.sh` from the `initContainers` section and to `release-license.sh` from the `preStop` hook.

If you prefer to register the server manually in the Solution Manager, then you can remove the `initContainers` section from the `denodo-service.yaml` file.

Once the scripts are saved in the right place and updated with the right paths, the cluster can be created with the following statements:

```
> kubectl create -f denodo-service.yaml
> kubectl autoscale deployment denodo-deployment --min=2 --max=10
```

Finally, due to the changes introduced in the RMI implementation of Denodo 8.0, it is necessary to adapt the YAML scripts to the new ports definition. So you will notice that below there are two different versions of the script *denodo-service.yaml*, one valid for Denodo 7.0 and another one for Denodo Platform 8.0. You should take the right script for the Denodo Platform running in your environment.

7 AUTOSCALING SCRIPTS

```
#!/bin/sh


if ! dpkg-query -W curl jq > /dev/null 2>&1; then
    apt update
    apt install -y curl jq
fi

type_node=VDP


curl --user $SOLUTION_MANAGER_USERNAME:$SOLUTION_MANAGER_PASSWORD --cookie-
jar ./cookie http://$SOLUTION_MANAGER_HOST:$SOLUTION_MANAGER_PORT/login


server_id=$(curl --cookie ./cookie --request GET http://
$SOLUTION_MANAGER_HOST:$SOLUTION_MANAGER_PORT/servers | jq -c --arg
SOLUTION_MANAGER_SERVER_NAME "$SOLUTION_MANAGER_SERVER_NAME" '.[] | .[] |
select (.name == $SOLUTION_MANAGER_SERVER_NAME) | { id: .id } ' | cut -b 7-
| sed 's/.$//')


# The script registers the server if it does not exist in the environment
if [ -z "$server_id" ]; then

     environment_id=$(curl --cookie ./cookie --header "Content-Type:
application/json" --request GET http://$SOLUTION_MANAGER_HOST:
$SOLUTION_MANAGER_PORT/environments | jq -c --arg
SOLUTION_MANAGER_ENVIRONMENT_NAME "$SOLUTION_MANAGER_ENVIRONMENT_NAME" '.[]
| select (.name == $SOLUTION_MANAGER_ENVIRONMENT_NAME) | { id: .id } ' | cut
-b 7- | sed 's/.$//')

     cluster_id=$(curl --cookie ./cookie --header "Content-Type:
application/json" --request GET http://$SOLUTION_MANAGER_HOST:
$SOLUTION_MANAGER_PORT/environments/$environment_id/clusters | jq -c --arg
SOLUTION_MANAGER_CLUSTER_NAME "$SOLUTION_MANAGER_CLUSTER_NAME" '.[] | .[] |
select (.name == $SOLUTION_MANAGER_CLUSTER_NAME) | { id: .id } ' | cut -b 7-
| sed 's/.$//')

     template='{"name": "%s", "clusterId": "%s", "typeNode": "%s", "urlIP":
"%s", "urlPort": "%s", "defaultDatabase": "%s", "username": "%s",
"password": "%s", "useKerberos": %s, "usePassThrough": %s } '

     create_server_json=$(printf "$template"
"$SOLUTION_MANAGER_SERVER_NAME" "$cluster_id" "$type_node" "$HOSTNAME"
"$VDP_REGISTRY_PORT" "$VDP_DEFAULT_DATABASE" "$VDP_USERNAME" "$VDP_PASSWORD"
"$SOLUTION_MANAGER_USE_KERBEROS" "$SOLUTION_MANAGER_USE_PASS_THROUGH" )
```

```
      curl  --cookie  ./cookie  --header  "Content-Type:  application/json"
--data "$create_server_json" --request POST http://$SOLUTION_MANAGER_HOST:
$SOLUTION_MANAGER_PORT/servers

fi
```

register-server.sh

```
#!/usr/bin/env bash

if ! dpkg-query -W curl > /dev/null 2>&1; then
    apt update
    apt install -y curl
fi


curl --user $SOLUTION_MANAGER_USERNAME:$SOLUTION_MANAGER_PASSWORD --cookie-
jar ./cookie http://$SOLUTION_MANAGER_HOST:$SOLUTION_MANAGER_PORT/login

curl          --request          GET          "http://$LICENSE_MANAGER_HOST:
$LICENSE_MANAGER_PORT/externalShutdown?
hostDomainName=$HOSTNAME&nodeIp=&port=$VDP_REGISTRY_PORT&product=VDP"
```

release-license.sh

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: server-config
  namespace: default
data:
  LICENSE_MANAGER_HOST: "license-manager"
  LICENSE_MANAGER_PORT: "10091"
  SOLUTION_MANAGER_HOST: "solution-manager"
  SOLUTION_MANAGER_PORT: "10090"
  SOLUTION_MANAGER_USERNAME: "username"
  SOLUTION_MANAGER_PASSWORD: "password"
  SOLUTION_MANAGER_ENVIRONMENT_NAME: auto-scaling-environment
  SOLUTION_MANAGER_CLUSTER_NAME: auto-scaling-cluster
  SOLUTION_MANAGER_SERVER_NAME: auto-scaling-server
  SOLUTION_MANAGER_USE_KERBEROS: "false"
  SOLUTION_MANAGER_USE_PASS_THROUGH: "false"
  VDP_USERNAME: "username"
  VDP_PASSWORD: "password"
  FACTORY_PORT: "8997"
  VDP_REGISTRY_PORT: "9999"
  VDP_DEFAULT_DATABASE: "vdb"
---
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
  sessionAffinity: ClientIP
---
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"]
        envFrom:
        - configMapRef:
            name: server-config
        ports:
        - name: jdbc-rmi-rgstry
          containerPort: 9999
        - name: jdbc-rmi-fctory
          containerPort: 8997
        - name: odbc
          containerPort: 9996
```

```
        - name: web-container
          containerPort: 9090
        lifecycle:
          preStop:
            exec:
              command: [sh, -c, "kubernetes/release-license.sh"]
        volumeMounts:
        - name: solution-manager-properties
          mountPath: /opt/denodo/conf/SolutionManager.properties
          readOnly: true
        - name: free-server-license
          mountPath: /opt/denodo/kubernetes/release-license.sh
      initContainers:
        - name: register-server
          image: everpeace/curl-jq
          command: [sh, -c, "opt/denodo/kubernetes/register-server.sh"]
          envFrom:
          - configMapRef:
              name: server-config
          volumeMounts:
          - name: register-server
            mountPath: /opt/denodo/kubernetes/register-server.sh
      volumes:
      - name: solution-manager-properties
        hostPath:
          path: /C/Kubernetes/SolutionManager.properties
      - name: register-server
        hostPath:
          path: /C/Kubernetes/register-server.sh
      - name: free-server-license
        hostPath:
          path: /C/Kubernetes/release-license.sh
```

denodo-service.yaml for Denodo 7.0

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: server-config
  namespace: default
data:
  LICENSE_MANAGER_HOST: "license-manager"
  LICENSE_MANAGER_PORT: "10091"
  SOLUTION_MANAGER_HOST: "solution-manager"
  SOLUTION_MANAGER_PORT: "10090"
  SOLUTION_MANAGER_USERNAME: "username"
  SOLUTION_MANAGER_PASSWORD: "password"
  SOLUTION_MANAGER_ENVIRONMENT_NAME: auto-scaling-environment
  SOLUTION_MANAGER_CLUSTER_NAME: auto-scaling-cluster
  SOLUTION_MANAGER_SERVER_NAME: auto-scaling-server
  SOLUTION_MANAGER_USE_KERBEROS: "false"
  SOLUTION_MANAGER_USE_PASS_THROUGH: "false"
  VDP_USERNAME: "username"
```

```
  VDP_PASSWORD: "password"
  FACTORY_PORT: "8995"
  VDP_REGISTRY_PORT: "9999"
  VDP_DEFAULT_DATABASE: "vdb"
---
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
  sessionAffinity: ClientIP
---
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"]
        envFrom:
        - configMapRef:
            name: server-config
        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:
          preStop:
            exec:
              command: [sh, -c, "kubernetes/release-license.sh"]
        volumeMounts:
        - name: solution-manager-properties
          mountPath: /opt/denodo/conf/SolutionManager.properties
          readOnly: true
        - name: free-server-license
          mountPath: /opt/denodo/kubernetes/release-license.sh
      initContainers:
        - name: register-server
          image: everpeace/curl-jq
          command: [sh, -c, "opt/denodo/kubernetes/register-server.sh"]
          envFrom:
          - configMapRef:
              name: server-config
          volumeMounts:
          - name: register-server
            mountPath: /opt/denodo/kubernetes/register-server.sh
      volumes:
      - name: solution-manager-properties
        hostPath:
          path: /C/Kubernetes/SolutionManager.properties
      - name: register-server
        hostPath:
          path: /C/Kubernetes/register-server.sh
      - name: free-server-license
        hostPath:
          path: /C/Kubernetes/release-license.sh
```

denodo-service.yaml for Denodo 8.0

8 REFERENCES

Denodo Knowledge Base: Deploying Denodo in Kubernetes
Kubernetes Documentation for Init Containers: Kubernetes Init Containers
Kubernetes Documentation for the Autoscaler: Kubernetes Horizontal Pod Autoscaler