Deploying Denodo in Amazon Elastic Container Service (ECS) using Fargate

Revision 20211122

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

Amazon Elastic Container Service (Amazon ECS) is a container management service that makes it easy to run, stop, and manage Docker containers on a cluster, letting you launch and stop container-based applications with simple API calls.

To use Amazon ECS you need a **cluster** that can be created within a new or an existing VPC. Then, after a cluster is up and running, the **task definitions** and **services** created will specify which Docker images run across the clusters. The **container images** are stored in and pulled from container registries, which may exist within the AWS infrastructure (Amazon ECR) or outside.

The services and tasks launched can use one of the following Amazon ECS launch types:

- **AWS Fargate**: The Fargate launch type allows running containerized applications without the need for provisioning or managing the backend infrastructure (Amazon EC2 instances). Just registering the task definition will make AWS Fargate launch the container.
- **Amazon EC2**: The EC2 launch type allows running containerized applications on a cluster of custom Amazon EC2 instances.

In this article, we will show how to run Docker images of Denodo on an Amazon ECS cluster using Fargate, for images stored in Amazon ECR

1 AWS COMMAND LINE INTERFACE

The AWS Command Line Interface (AWS CLI) is an open-source tool that can be used for administering the AWS 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.

In addition to using the AWS CLI to access Amazon ECS resources, you can use the Amazon ECS CLI, which provides high-level commands to simplify creating, updating, and monitoring clusters and tasks from a local development environment using Docker Compose.

Although it is also possible to launch our first cluster using the Amazon ECS first-run wizard, in this article we have chosen to use the **AWS CLI** because usually, the commands are self-explanatory and also because it is possible to use them and test them easily in any custom AWS deployment.

2 AMAZON ECS USING FARGATE

AWS Fargate is a compute engine for containers that can be used with Amazon ECS to run containers without having to manage servers or clusters of Amazon EC2 instances.

When running the ECS tasks and services with the Fargate launch type, Fargate will read the specifications from the configuration files and the containerized applications will be launched following the CPU and memory requirements specified. Notice that each Fargate task has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.

2.1 LIST OF PREREQUISITES

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

- A valid AWS account with enough privileges to build all the AWS elements involved in the deployment of the Denodo image.
- An **IAM entity**, to avoid accessing AWS directly with the credentials of our AWS account. With the AWS Identity and Access Management (IAM) service, we can have restricted permissions and policies over the AWS resources.
- A task execution IAM role (`ecsTaskExecutionRoleARN`) that provides permissions to Amazon ECS container agents to use the private registry authentication feature to pull the container image from Amazon ECR. It requires an IAM policy and role for the service to know that the agent belongs to you.
- A Virtual Private Cloud (Amazon VPC) with public and private subnets (a list of subnet ids separated by commas - `subnetIdList`) for our clusters and a dedicated security group (`securityGroupId`). We want to expose the Denodo Server to the internet so in this guide we will be using a public subnet and assign a public IP to the ECS task, but you could use a private subnet configured with a NAT waterway with an elastic IP address.
- The latest version of the AWS CLI, because although the AWS Management Console can be used to manage Amazon ECS, the **AWS CLI** allows you to build scripts that can automate common management tasks in Amazon ECS.
- A locally accessible Docker image of Denodo. If you don't have one you can check how to create your own Docker images for Denodo Platform Containers

2.2 AWS PERMISSIONS

An IAM entity (user, group, or role) must be set up with the privileges required by the AWS CLI for running the statements in this guide.

In this article, we have configured an IAM Role (`roleARN`) to use ECS and Fargate/EC2 services.

This IAM role must be able to list, push, and pull images in the ECR repository, which is granted with the following AWS managed policy:

- `AmazonEC2ContainerRegistryPowerUser`.

To grant someone permission to create an Amazon ECR repository with the Amazon ECR CreateRepository API operation, we must include the following action in their policy:

- `ecr:CreateRepository`

This IAM role must be able to list, push, and pull images in the ECR repository, which is granted with the following AWS managed policy:

- `ecsTaskExecutionRole`

In addition, although this works automatically with the cluster creation, notice that Amazon ECS uses the service-linked role named `AWSServiceRoleForECS` to enable Amazon ECS to call AWS APIs on your behalf.

2.3 CONFIGURE THE AWS CLI

To start with the AWS configuration, open a new console and execute the below AWS commands that will do the basic configuration of the AWS CLI.

Configure the AWS access keys for the AWS account by using the following command:

```
$ aws configure
```

When entering this command, the AWS CLI prompts for the following information:

- AWS Access Key ID
- AWS Secret Access Key
- Default region name
- Default output format

To read more about the AWS CLI configuration check the AWS CLI user guide.

At this point, you need to decide if you will configure the default AWS CLI profile or if you want to create a custom profile. If you create a custom profile some of the commands in the rest of the guide will require an additional parameter to specify the profile to use (`--profile profilename`) so for simplicity, we will just configure the default profile in this guide. You can read more about named profiles in the AWS Command Line Interface documentation.

Use these commands to configure the IAM role in the default profile:

```
$ aws configure set role_arn <roleARN>
$ aws configure set source_profile default
```

The **role_arn** parameter specifies the Amazon Resource Name (ARN) of an IAM role that you want to use to perform operations with the default profile. It will have this format: `arn:aws:iam::<accountNumber>:role/<roleName>`

The **source_profile** parameter identifies what profile to use to find credentials that have permission to assume the IAM role.

2.4 CREATE THE CONTAINER REGISTRY

Create a new Elastic Container Registry by providing a custom name for the repository:

```
$ aws ecr create-repository --repository-name <denodo-ecr>
```

Check that the `denodo-ecr` repository has been created successfully and obtain its `repositoryUri` from the output of the `create-repository` statement. The field `repositoryUri` will be used later to authenticate the local Docker client.

To obtain the `repositoryUri` value the following command can be used:

```
$ aws ecr describe-repositories --repository-name <denodo-ecr> --query
"repositories[].repositoryUri"
```

Authenticate the Docker client to the new registry to push the Denodo image to the repository using the `repositoryUri`:

```
$ aws ecr get-login-password | docker login --username AWS --password-stdin
<repositoryUri>
```

Finally, tag the Denodo image and push it to the Elastic Container Registry

```
$ docker tag denodo-platform:latest <repositoryUri>
$ docker push <repositoryUri>
$ aws ecr list-images --repository-name <denodo-ecr>
```

2.5 CREATE THE ECS CLUSTER

To deploy the Denodo image on ECS it is necessary to create a cluster. By default, the AWS accounts have a default cluster. The benefit of using this default cluster is that you don't have to specify the `--cluster <cluster-name>` option in the subsequent commands. But if you do create your own cluster, then you must specify `--cluster <cluster-name>` for each command that you intend to use with that cluster.

```
$ aws ecs create-cluster --cluster-name <denodo-ecs-cluster>
```

It can take some time for the cluster to be available, but it is possible to check its status with the following command:

```
$ aws ecs describe-clusters --cluster <denodo-ecs-cluster>
```

2.6 REGISTER AN ECS TASK

Once the cluster is created, you can start registering the **task definitions** that will be run as tasks on the new cluster created.

This example uses a simple task definition named `denodo-task-definition` that makes use of the Denodo container image hosted on the new repository `denodo-ecr`.

Also, notice that the Denodo Platform requires a valid license to run, which can be obtained from a *Denodo License Manager server* or in some scenarios, like evaluation or Denodo Express, as a *standalone license* file:

- If you intend to use a license managed by the License Manager you need an appropriate `SolutionManager.properties` configuration file that points to the License Manager server.
- Alternatively, you can use a Denodo standalone license, if available.

In both scenarios, the Denodo server will expect a file in the `<DENODO_HOME>/conf` directory, either `SolutionManager.properties` or `denodo.lic`. The files could be added to the image, but in order to make this configuration dynamic, we will load the files from the task definition.

Hence, the solution proposed is based on the ability of the task definitions to specify environment variables, which can be used later to create files. Therefore, we can define an environment variable, for instance with the content of the `SolutionManager.properties` in `base64` (which is more convenient than working with real text in the task definition) and materialize that variable value with the following command before starting the server:

```
echo $SOLUTION_MANAGER_CONF_B64 | base64 -d > /opt/denodo/conf/SolutionManager.properties;
```

Also, the task definition can be specified in a JSON file, which improves the readability and the management of the tasks because these definitions can be very large. Below you will find two examples of very simple task definitions, one per each type of license scenario.

Note: As AWS Fargate is a serverless compute engine, the characteristics of the server that will run the container are not known in advance. To be able to run Denodo with a core based license the number of cores used by the container needs to be limited. To do so, the "cpu" option can be used in the Denodo task definition configuration files.

For more information see Task definition parameters - Amazon Elastic Container Service.

2.6.1 Solution Manager

If there is a Denodo License Manager server running and accessible by the cluster then its connection details can be configured in the Solution Manager Configuration file.

As explained above, the task definition includes the content of the `SolutionManager.properties` configuration file encoded using `base64`. It is necessary to replace the `SOLUTION_MANAGER_CONF_B64` placeholder in the task definition with the `base64` encoding of the file. Note that you can get the `base64` value with a command available in your OS or using an online service. So for instance, if you have the following `SolutionManager.properties` file:

```
# License Manager Configuration
com.denodo.license.host=solution-manager
com.denodo.license.port=10091
com.denodo.license.maxRetries=3
com.denodo.license.millisecondsBetweenRetries=5000
```

`SolutionManager.properties`

The equivalent `base64` encoding would be:

```
IyBMaWNlbnNlIE1hbmFnZXIgQ29uZmlndXJhdGlvbgpjb20uZGVub2RvLmxpY2Vuc2Uua
G9zdD1zb2x1dGlvbi1tYW5hZ2VyCmNvbS5kZW5vZG8ubGljZW5zZS5wb3J0PTEwMDkxCm
NvbS5kZW5vZG8ubGljZW5zZS5tYXhSZXRyaWVzPTMKY29tLmRlbm9kby5saWNlbnNlLm1
pbGxpc2Vjb25kc0JldHdlZW5SZXRyaWVzPTUwMDAK
```

`SolutionManager.properties` in `base64`

At this point, you are ready to register the task definition, so take the following template and replace the placeholders `SOLUTION_MANAGER_CONF_B64`, `repositoryUri`, and `ecsTaskExecutionRoleARN` to start a Denodo container in the ECS cluster:

```
{
   "containerDefinitions": [
      {
         "environment": [
           {
             "name": "SOLUTION_MANAGER_CONF_B64",
             "value": "<SOLUTION_MANAGER_CONF_B64>"
           }
         ],
         "entryPoint": [
           "sh",
           "-c",
                        "echo $SOLUTION_MANAGER_CONF_B64 | base64 -d >
/opt/denodo/conf/SolutionManager.properties;     ./denodo-container-start.sh
```

```
--vdpserver"
         ],
         "essential": true,
         "image": "<repositoryUri>:latest",
         "name": "denodo",
         "portMappings": [
            {
               "containerPort": 9999,
               "hostPort": 9999,
               "protocol": "tcp"
            },
            {
               "containerPort": 9996,
               "hostPort": 9996,
               "protocol": "tcp"
            }
         ]
      }
   ],
   "executionRoleArn": "<ecsTaskExecutionRoleARN>",
   "family": "denodo-task-definition",
   "networkMode": "awsvpc",
   "memory": "8192",
   "cpu": "4096",
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

`denodo-task-definition.json` with Solution Manager

Once the task definition is saved to the JSON file `denodo-task-definition.json` you can create the task definition with the following command:

```
$ aws ecs register-task-definition --cli-input-json file://denodo-task-definition.json
```

Then, you can list the task definitions created with the following command:

```
$ aws ecs list-task-definitions
```

You can also check the details of a task definition:

```
$ aws ecs describe-task-definition --task-definition <taskDefinitionArn>
```

2.6.2 Standalone Licenses

To use a standalone license file, we are going to use the same solution as with the `SolutionManager.properties` file, but instead of defining an environment variable for the Solution Manager configuration file, it will be created for the license file.

The following template uses the environment variable `LICENSE_B64` to create the license file in `/opt/denodo/conf/denodo.lic`. You just need to get the base64 encoding for your license file and replace `<LICENSE_B64>` with it:

```
{
   "containerDefinitions": [
      {
         "environment": [
           {
             "name": "LICENSE_B64",
             "value": "<LICENSE_B64>"
           }
         ],
         "entryPoint": [
           "sh",
           "-c",
              "echo $LICENSE_B64 | base64 -d > /opt/denodo/conf/denodo.lic;
./denodo-container-start.sh --vdpserver"
         ],
         "cpu": "4096",
         "essential": true,
         "image": "<repositoryUri>:latest",
         "name": "denodo",
         "portMappings": [
            {
               "containerPort": 9999,
               "hostPort": 9999,
               "protocol": "tcp"
            },
            {
               "containerPort": 9996,
               "hostPort": 9996,
               "protocol": "tcp"
            }
         ]
      }
   ],
   "executionRoleArn": "<ecsTaskExecutionRoleARN>",
   "family": "denodo-task-definition",
   "networkMode": "awsvpc",
   "memory": "8192",
   "cpu": "4096",
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

`denodo-task-definition.json` with standalone licenses

2.7 CREATE AN ECS SERVICE

After registering the task, the next step is creating the Amazon ECS service that will run the task on our cluster. In this case, the service will run a single instance of the task registered. Note that the task requires a route to the internet, as we want the Denodo server to be accessed from outside AWS:

```
$ aws ecs create-service \
      --service-name <denodo-service> \
      --cluster <denodo-ecs-cluster> \
      --task-definition denodo-task-definition \
      --desired-count 1 \
      --launch-type "FARGATE" \
      --network-configuration
"awsvpcConfiguration={subnets=<subnetIdList>,securityGroups=<securityGroupId
>,assignPublicIp=ENABLED}" \
      --load-balancers
"targetGroupArn=<nlbARN>,containerName=denodo,containerPort=9999"
```

The `create-service` command returns a description of the task definition after it completes its registration. The following command gets a list of services created in the cluster:

```
$ aws ecs list-services --cluster <denodo-ecs-cluster>
```

You can get the description of the service running using the service name retrieved earlier to get more information about the task:

```
$ aws ecs describe-services --cluster <denodo-ecs-cluster> --services
<denodo-service>
```

2.8 CONNECTING TO DENODO PLATFORM

Finally, in order to connect to the server from a VDP client, you will need to ensure that the VDP client is able to resolve this hostname to the public IP address of the ECS service, and you can do that by adding an entry in the local `hosts` configuration file to map that hostname with the public IP address.

You can get the Public IP of the Task in the AWS Management Console or obtain it from the AWS CLI with the following process:

1. First, we will start obtaining the Task ARN that is running Denodo.
2. Then, with the Task we can get the Elastic Network Interface assigned to it, and
3. Finally we can retrieve the Public IP address from the ENI:

```
$ aws ecs list-tasks --cluster <denodo-ecs-cluster> --query "taskArns[*]"

$ aws ecs describe-tasks --cluster <denodo-ecs-cluster> --tasks <taskARN>
--query "tasks[*].attachments[*].details[1].value"
```

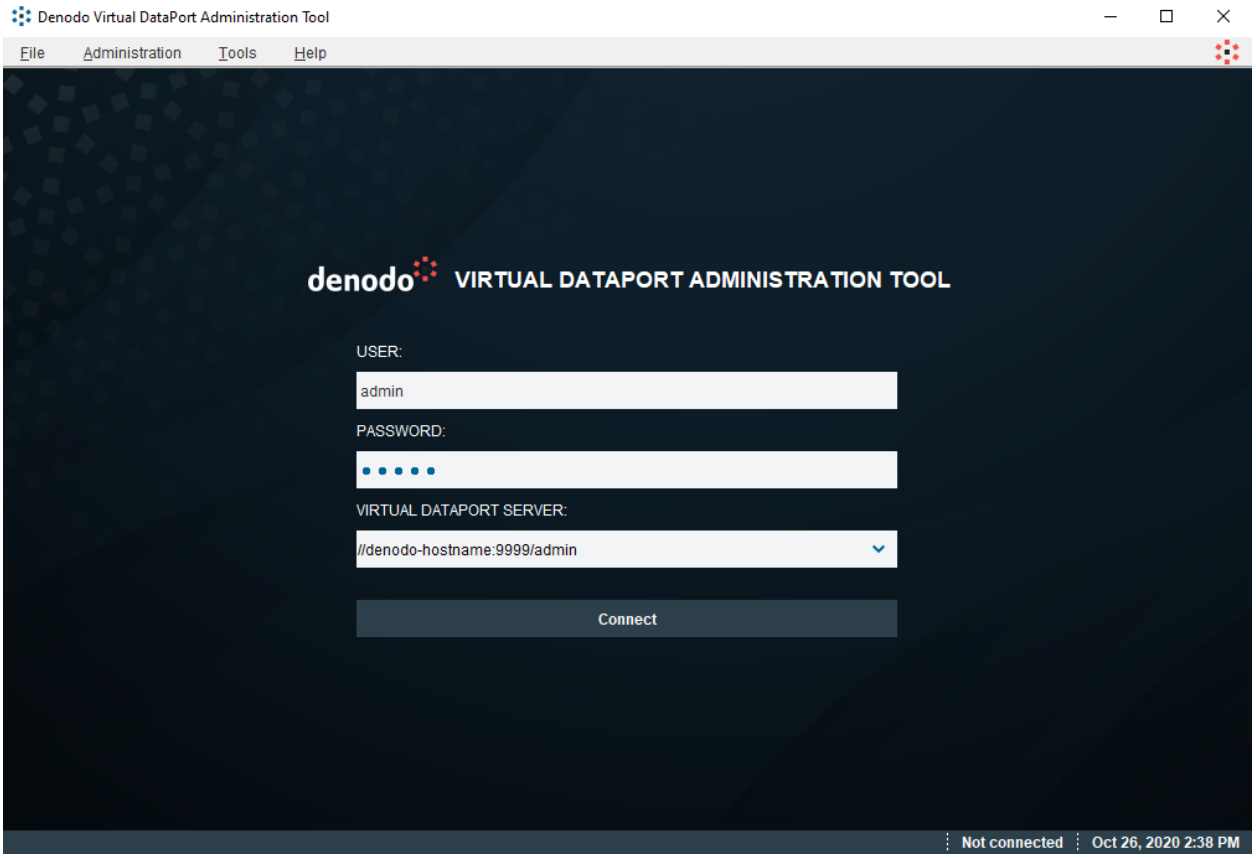
```
$ aws ec2 describe-network-interfaces --network-interface-ids <eni> --query
"NetworkInterfaces[*].Association.PublicIp"
```

Then, add the entry with the AWS Public IP address in the local `hosts` file:

```
# Denodo ECS service
<public-ip>        <denodo-hostname>
```

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

```
//<denodo-hostname>:9999/admin
```



Virtual DataPort Administration Tool

3 REFERENCES

AWS Command Line Interface
How to create your own Docker images for Denodo Platform Containers