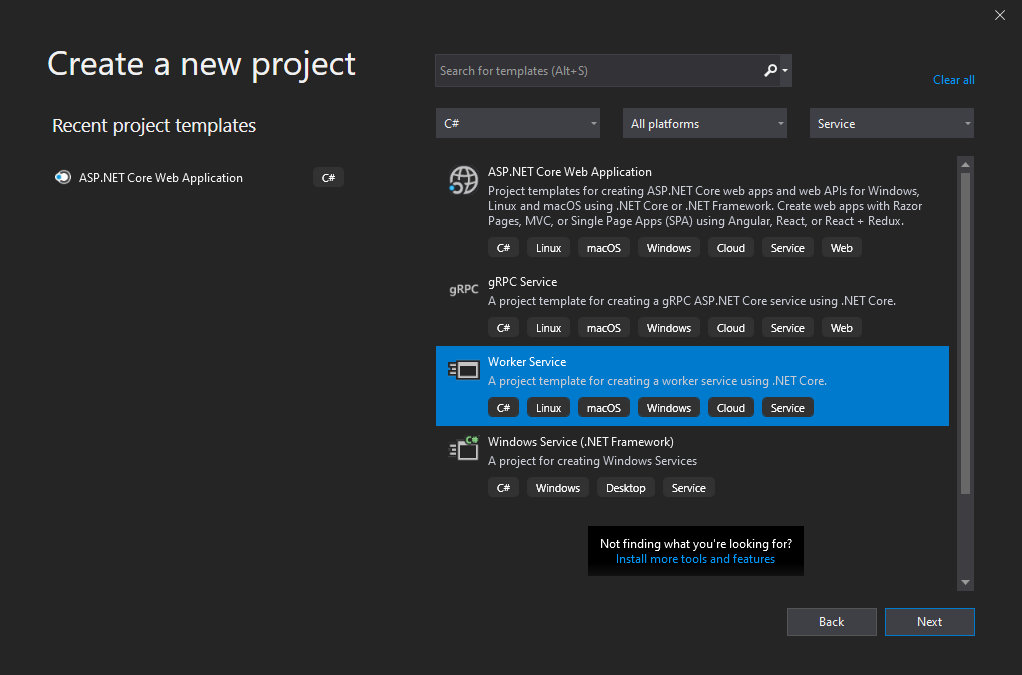
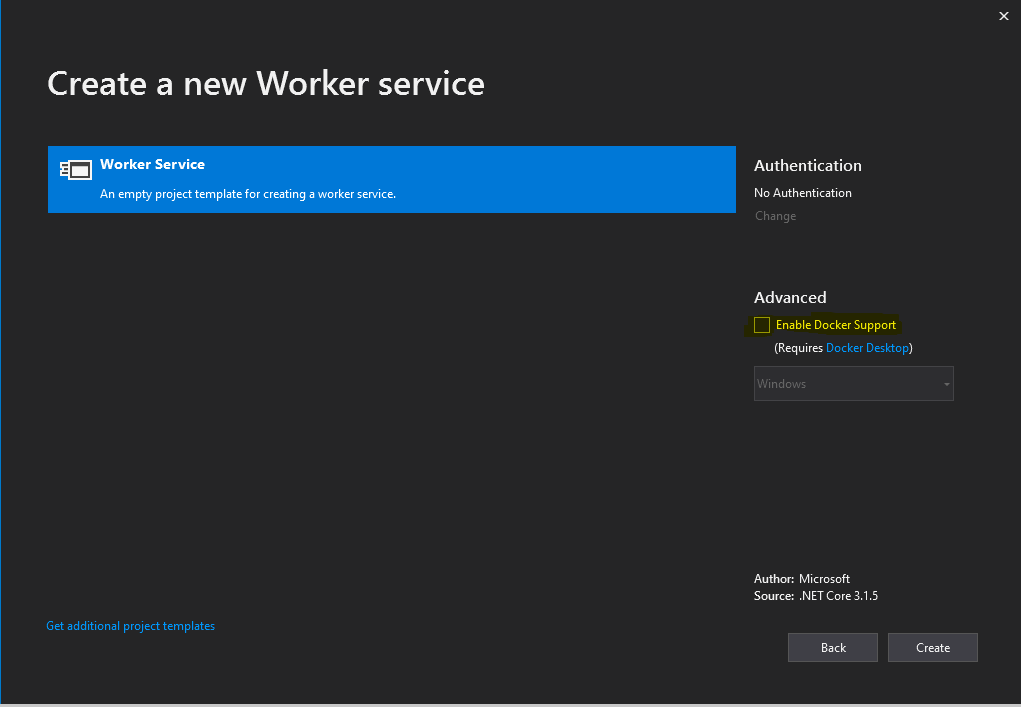
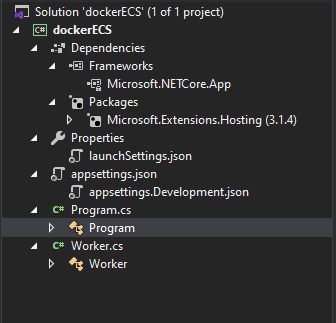
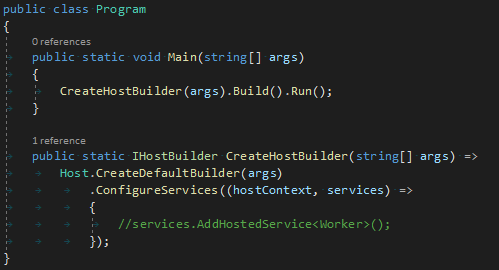
# Prerequires:

Visual Studio 2019, SQL Server 2016, .Net Core 3.1, Docker Desktop

# .Net Core Console Application

I have chosen .NET Core 3.1 Worker Service template for my application. Worker template as all the necessary component for a basic service. As this is a test application, I have disabled the background task.

Follow below steps,

1. Create new project by selecting worker service template  
   
2. We will be adding docker support manually, so do not tick “Enable Docker Support”  
   
3. Your project should have below files now.  
   
4. Let’s disabled the worker background task by commenting service injection ConfigureServices() method.  
   
5. Build and run your application to make sure that there is no error.
6. Create a new database “dockerECS” and a table “Name”  
   USE [dockerECS]  
   GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Name](

[id] [int] IDENTITY(1,1) NOT NULL,

[Name] [nchar](10) NULL

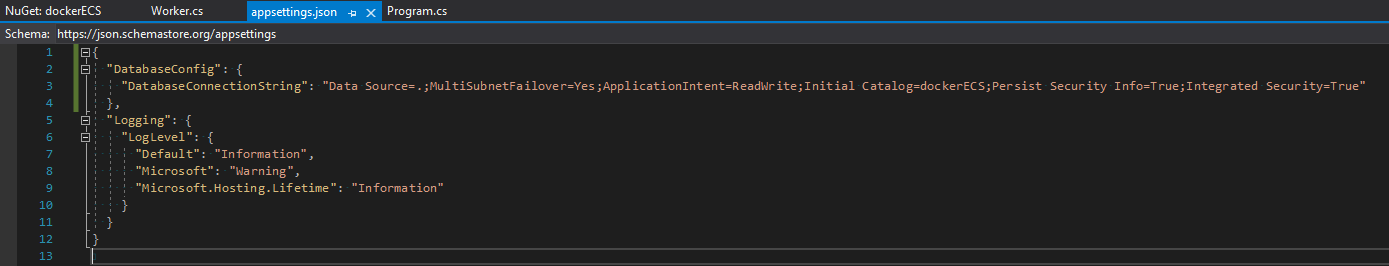
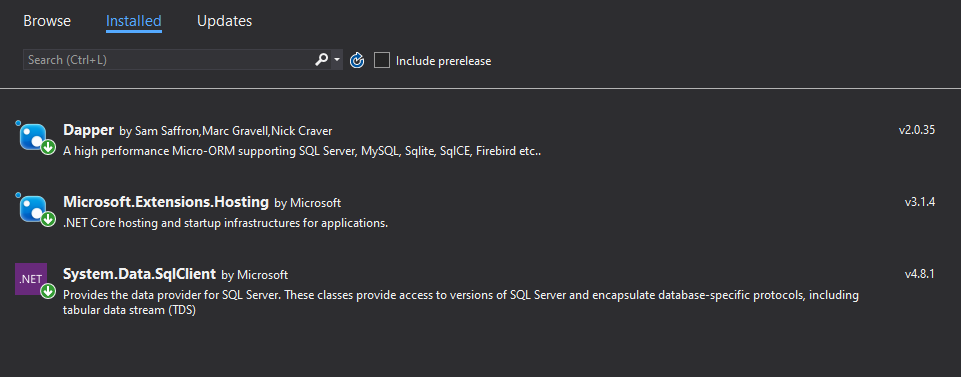
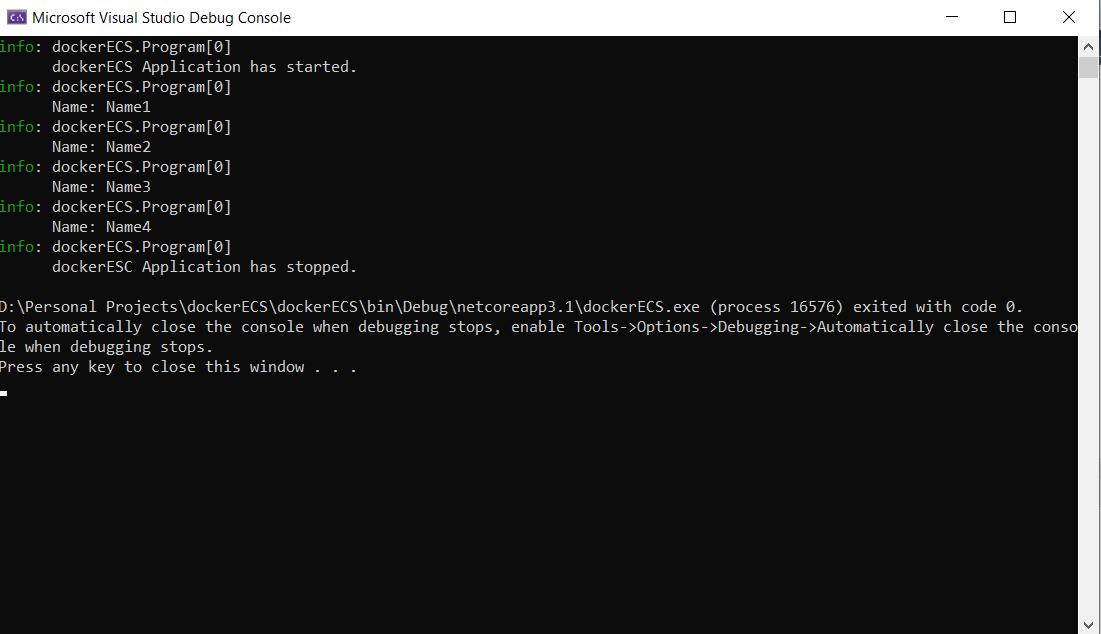
) ON [PRIMARY]

GO

INSERT INTO [dbo].[Name] (Name)

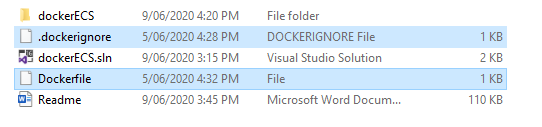
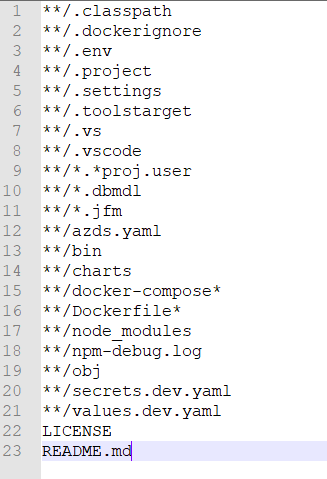
VALUES ('Name1'), ('Name2'), ('Name3'), ('Name4')

GO

1. Let’s add database connection to .NET core application. Add “DatabaseConfig” entry to appsettings.json  
   
2. I will be using dapper library to read data from the database, so install dapper from NuGet Package Manager. Also install System.Data.SqlClient.  
   
3. Now let’s modify Program.cs to read data from the table.  
   
4. Build and run the application, should see an output like this.  
   

# Containerize .Net Core application

We can now containerize our application and test it locally. I’ll be using linux containers.

1. Create a “Dockerfile” and “.dockerignore” files in the same folder as the solution file.  
   
2. Add following to docker ignore file, modify the list as required.  
   
3. Let’s build the docker file now. We first need to download the linux OS, I will be using Focal 3.1.4  
   FROM mcr.microsoft.com/dotnet/core/runtime:3.1.4-focal AS base

WORKDIR /app

1. Then we need to download .NET Core SDK 3.1  
   FROM mcr.microsoft.com/dotnet/core/sdk:3.1-focal AS build
2. We will create a source folder and copy our solution and project files.  
   WORKDIR /src

COPY \*.sln .

COPY ["dockerECS/\*.csproj", "dockerECS/"]

1. Now we need to restore necessary dependency  
   RUN dotnet restore
2. Let’s copy the solution and build the application.  
   COPY . .

RUN dotnet build

1. We can now build the release and publish versions  
   FROM build AS release

WORKDIR "/src/dockerECS"

RUN dotnet build "dockerECS.csproj" -c Release -o /app/build

FROM build AS publish

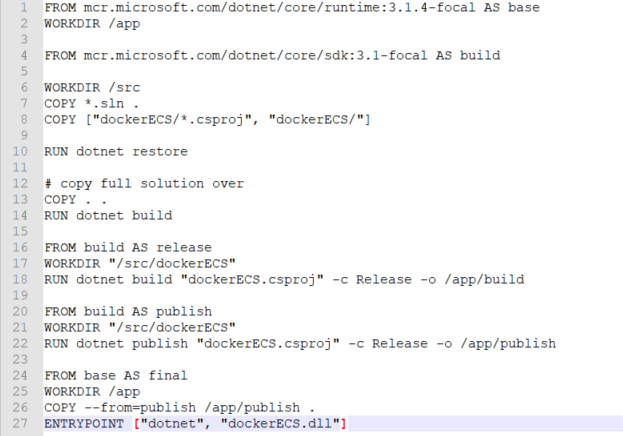
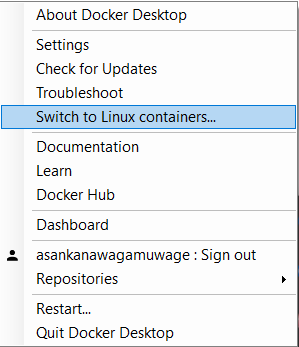
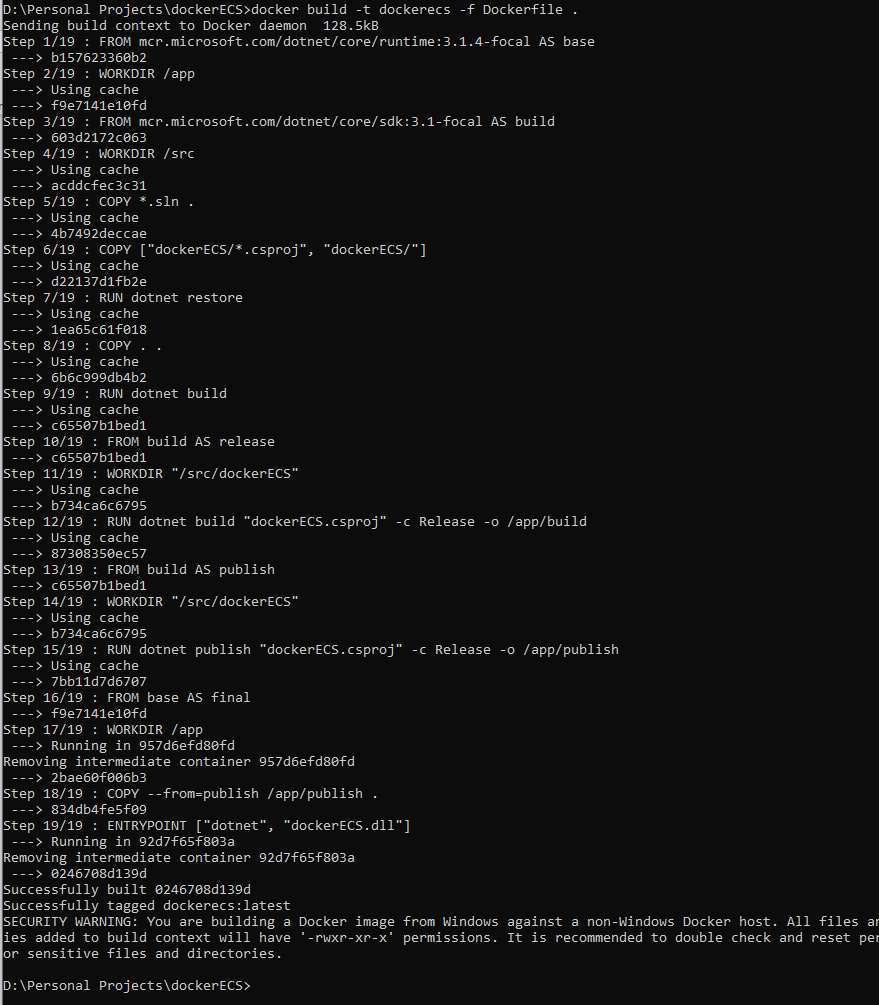
WORKDIR "/src/dockerECS-v3"

RUN dotnet publish "dockerECS.csproj" -c Release -o /app/publish

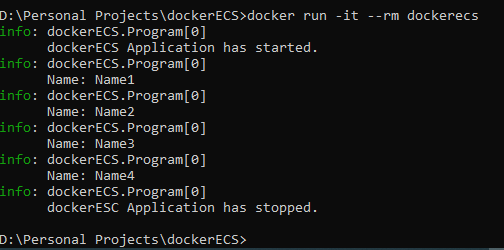
1. We now need to copy the dll from publish folder to our application folder  
   WORKDIR /app

COPY --from=publish /app/publish .

ENTRYPOINT ["dotnet", "dockerECS.dll"]

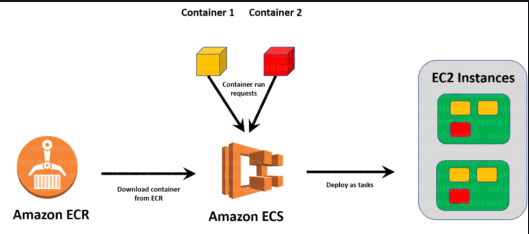
1. Final Dockerfile should be as below,  
   
2. Now we can build the docker image using this file, as we are using linux container change your docker desktop to linux containers.   
   
3. Now in your command prompt navigate to the folder where the dockerfile is located. Run below command to build the docker image.   
   docker build -t dockerecs -f Dockerfile .
4. You will see an output like this  
   
5. Now we should be able to run the application in a container using this image, but the application will fail to execute sql queries. When we run an application in a container, the sql server is no longer a local service. Therefore we cannot use the same connection string that we used before. We need to add a inbound rule for SQL and configure the connection string using docker NAT IP address. Please follow the instruction [here](https://vivekcek.wordpress.com/2018/06/10/connecting-to-local-or-remote-sql-server-from-docker-container/)

Docker won’t be able to use windows authentication, so create a username/password or use existing user  
  
"Data Source=<Your IP>\\SQLEXPRESS,1433;MultiSubnetFailover=<Your Port>;ApplicationIntent=ReadWrite;Initial Catalog=dockerECS;Persist Security Info=True;User ID=<User name>;Password=<Password>;MultipleActiveResultSets=true;"

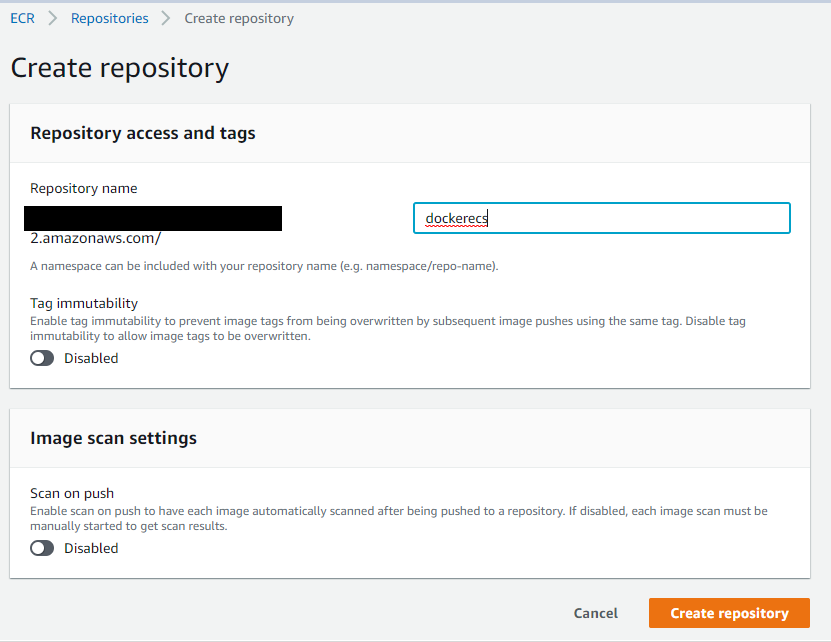
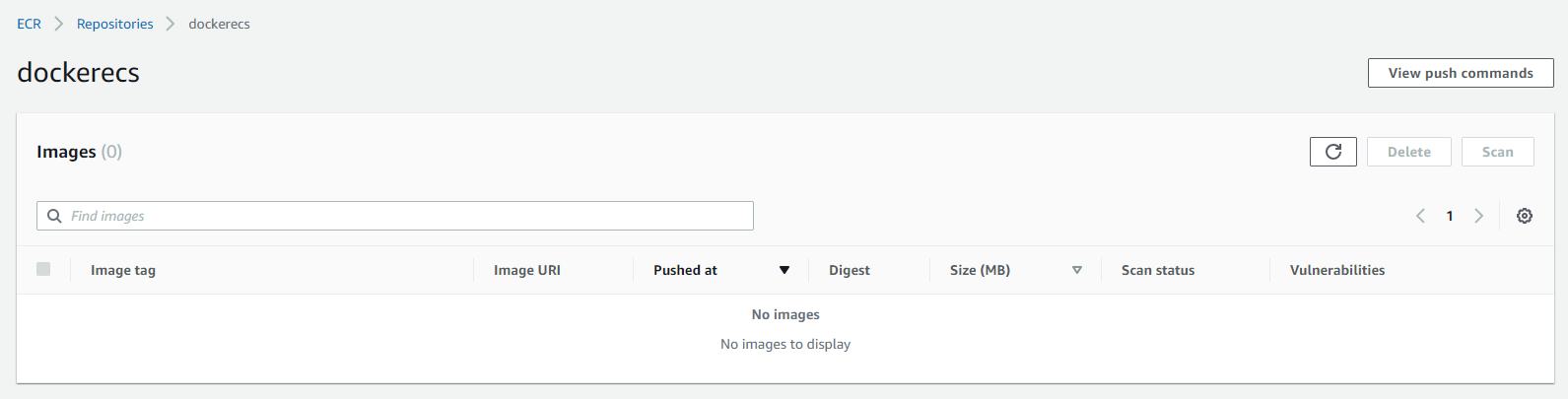
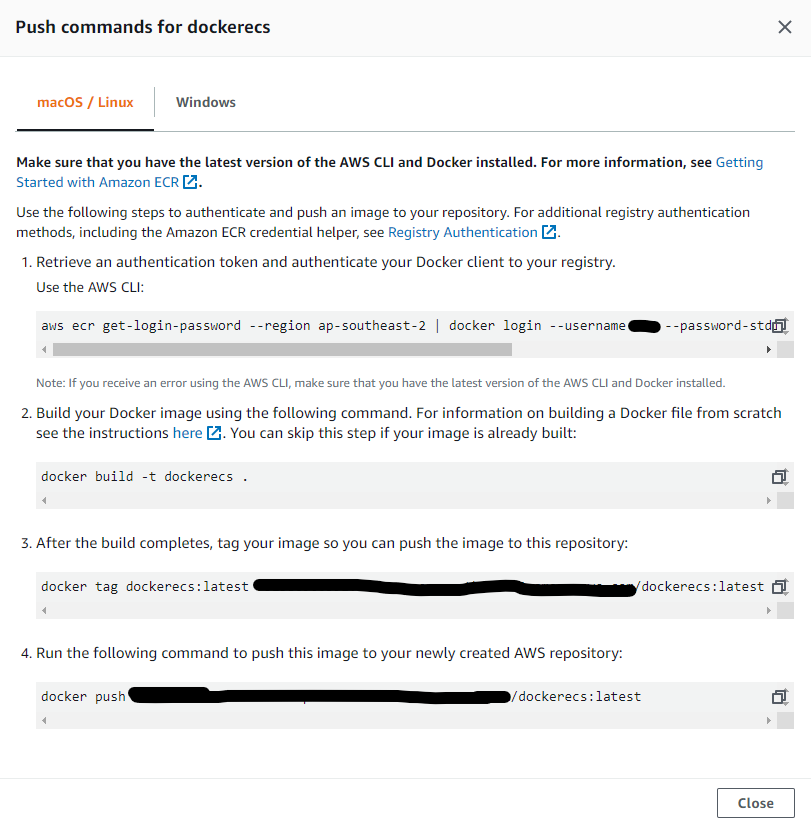
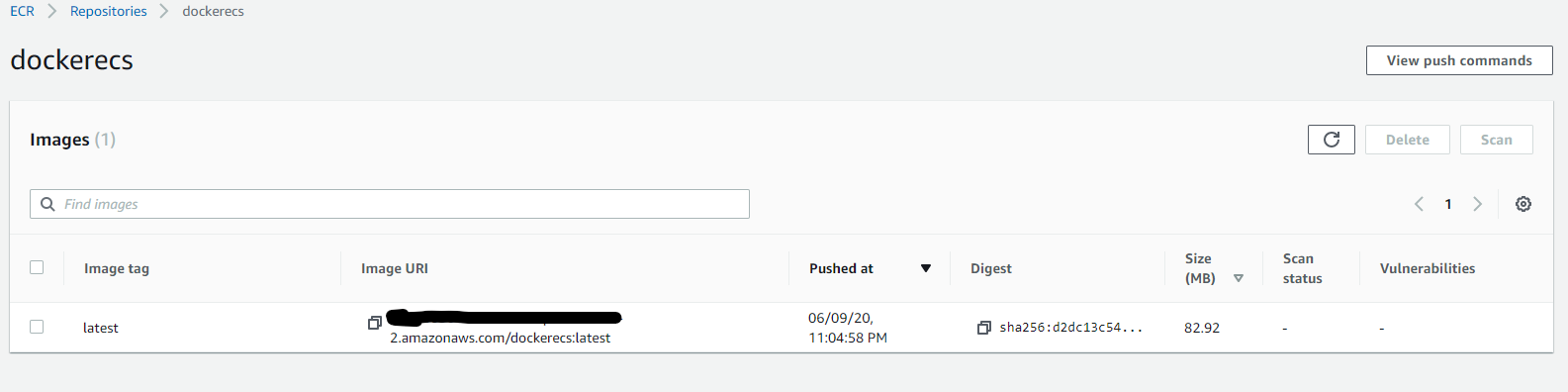
1. Build the image again using above command and run this command to start the container.  
   docker run -it --rm dockerecs  
   

# Deploy the application to AWS ECS

When we run docker application in AWS ECS, it needs to know from where it can download the docker image, this can be any repository service for docker images, but here we will be using AWS ECR. Below image explains the architecture.

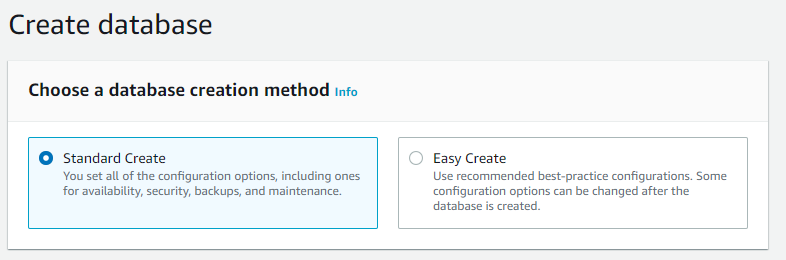
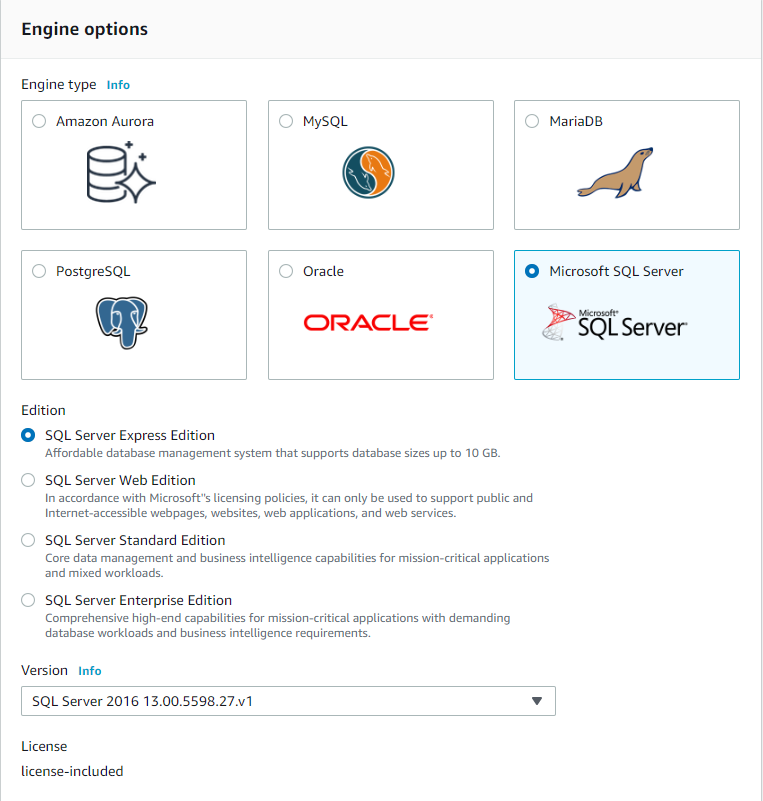


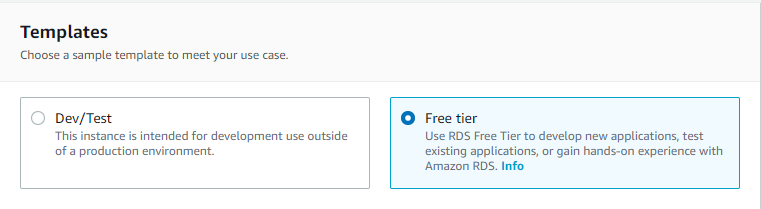
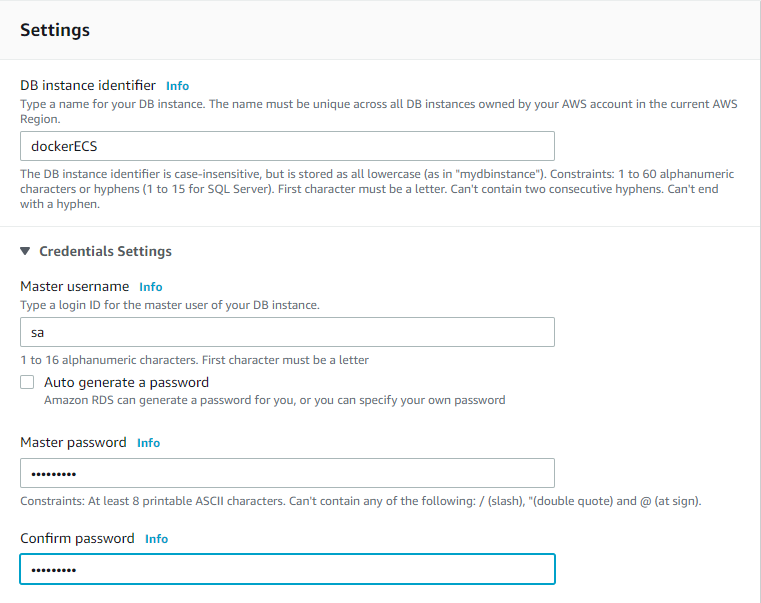
## 3.1 New repository in ECR

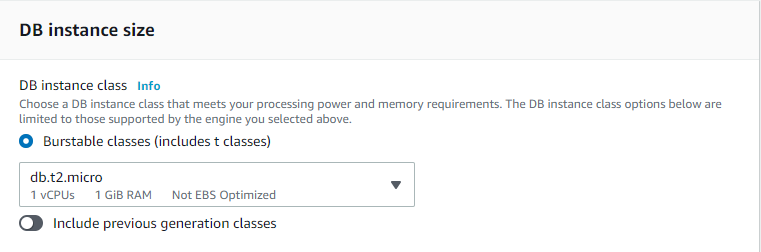
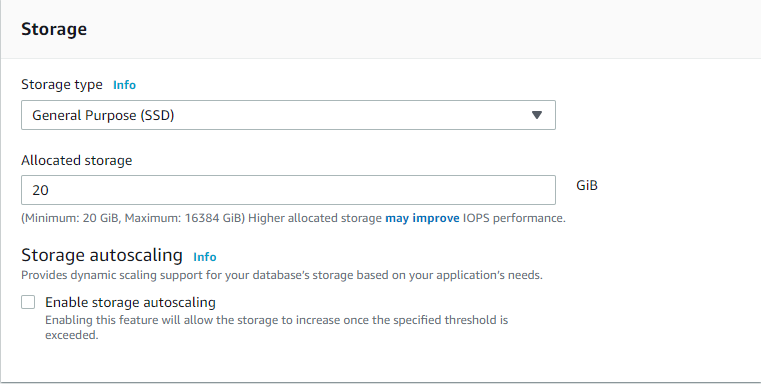
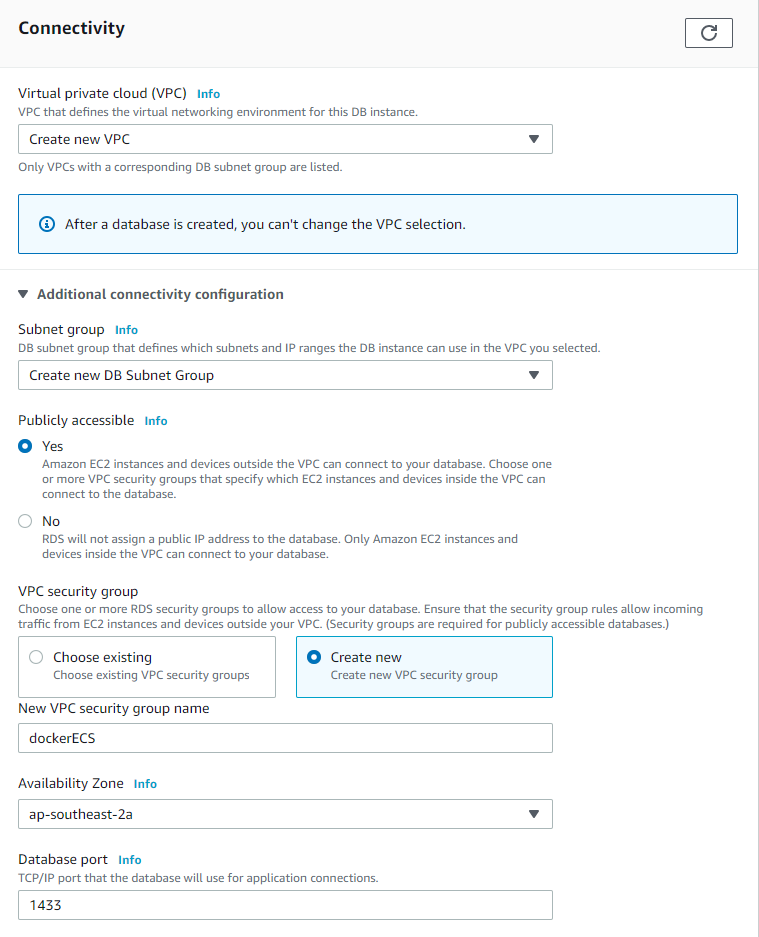
1. Let’s go to AWS console and create a repository in ECR.  
   AWS Console 🡪 ECR 🡪 Create Repository  
   
2. Type a name for repository and “Create Repository”
3. Now select the repository from the ECR list and hit “View push commands” button.  
   
4. Follow the instruction on the popup to push your local docker image to AWS  
   
5. Refresh the page and you will be able to see your docker image.  
   

## 3.2 Create a RDS Database

1. Go to AWS Console 🡪 RDS 🡪 Databases and click “Create database”
2. Choose following options and click “Create database” at the end

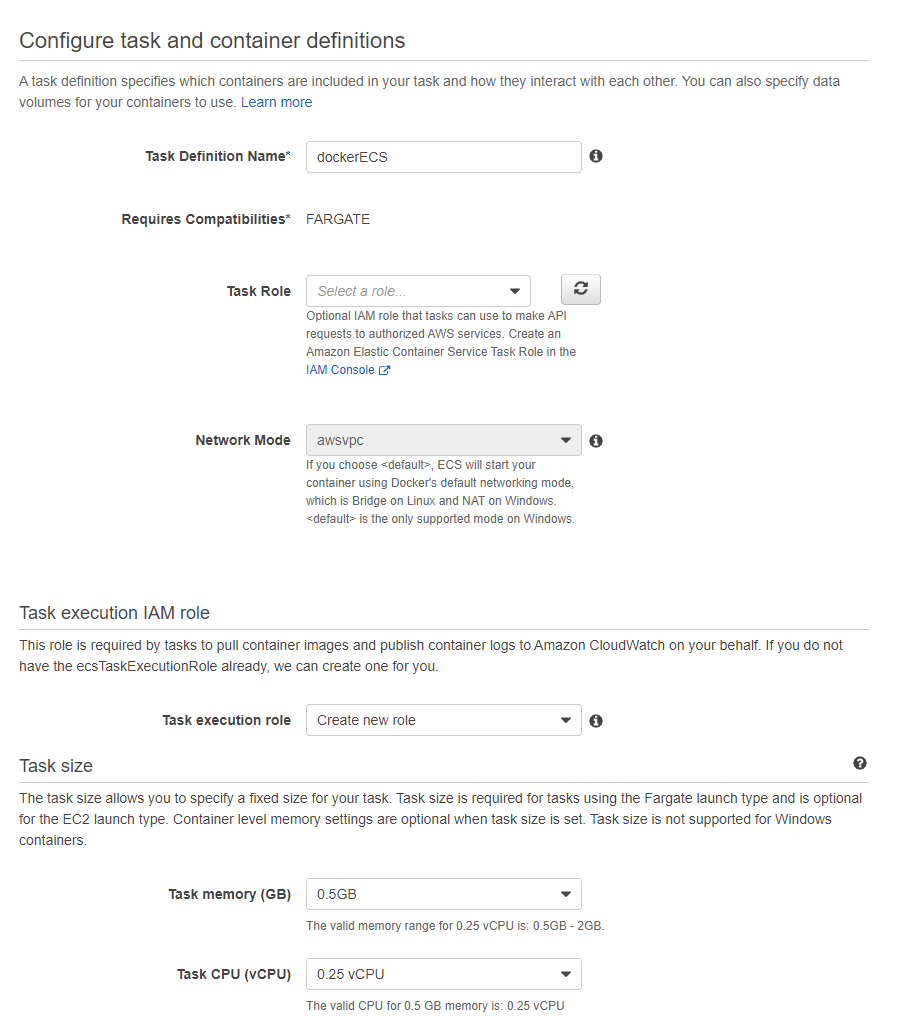
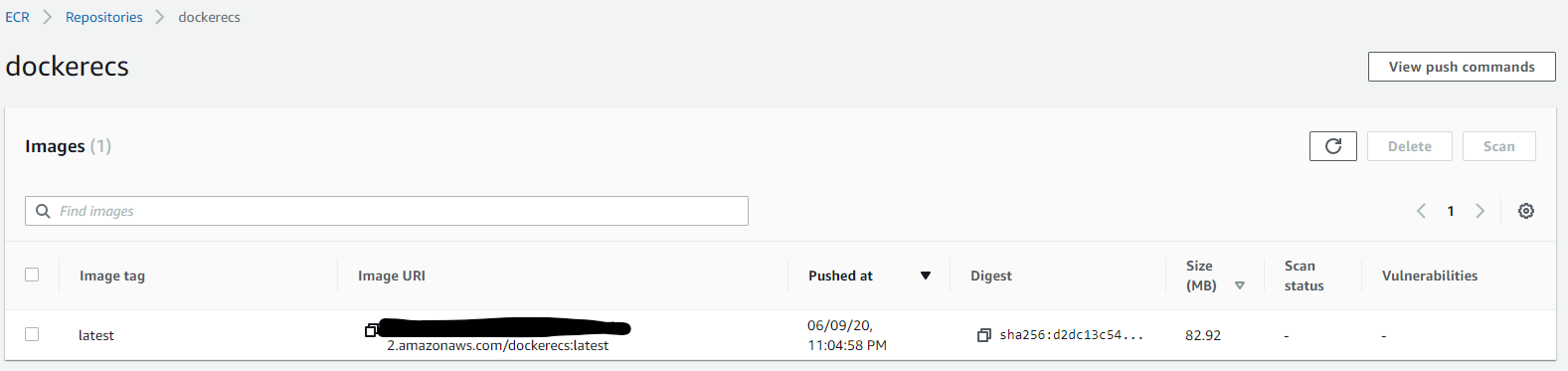
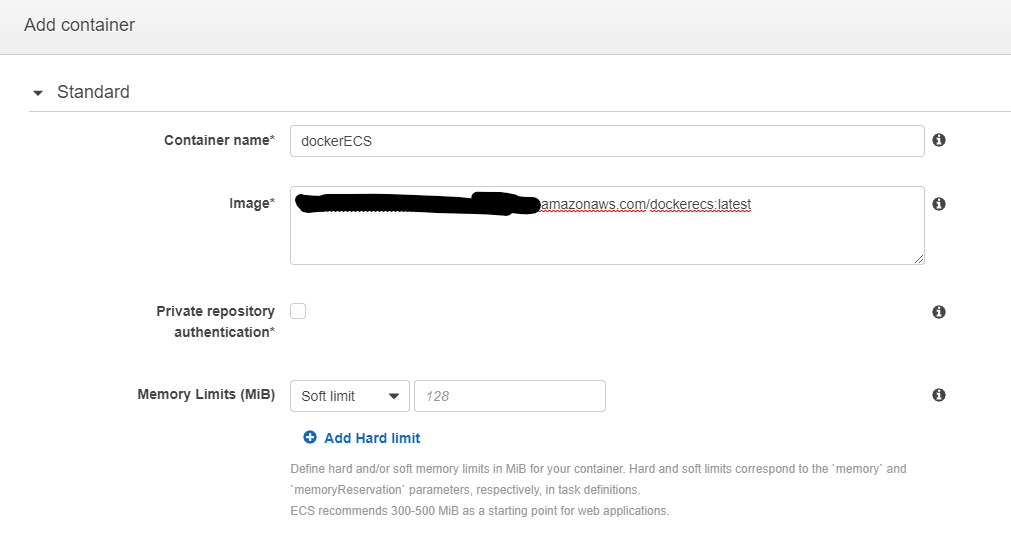
  
  


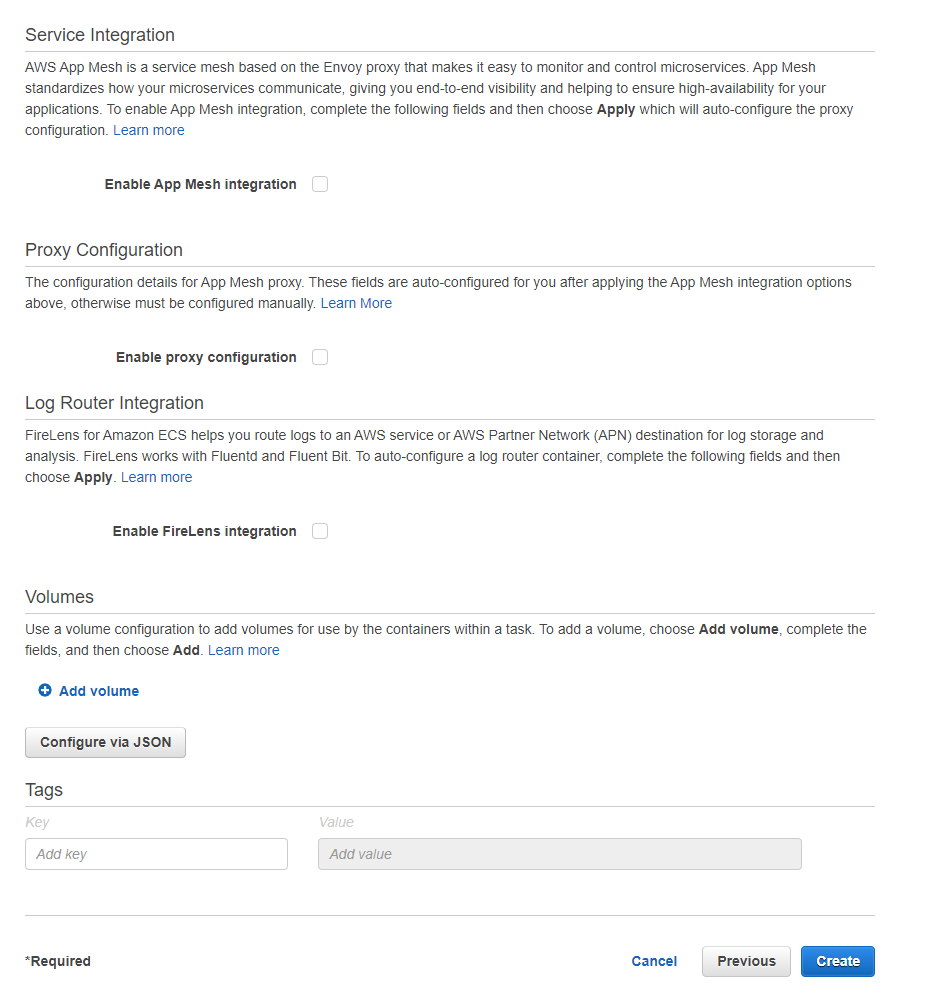
  
  


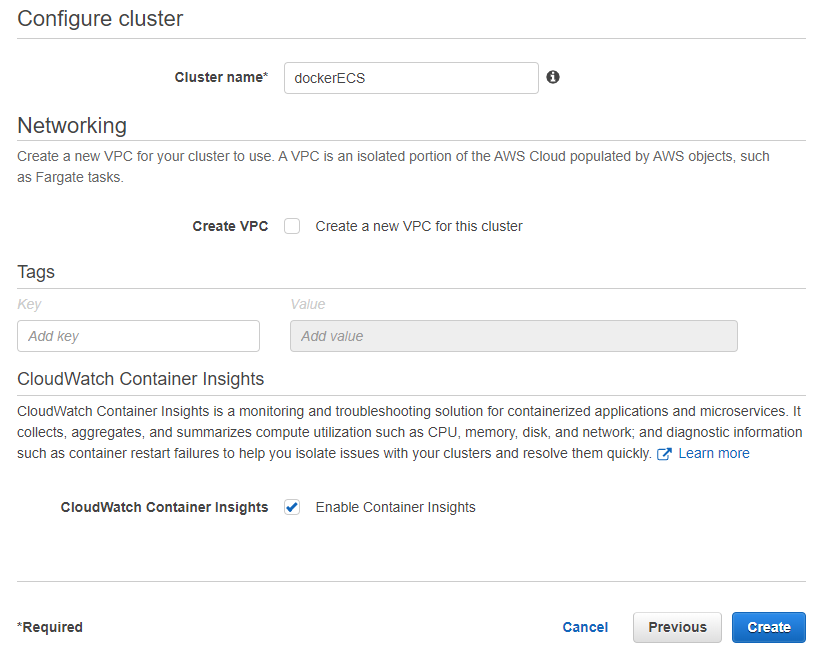
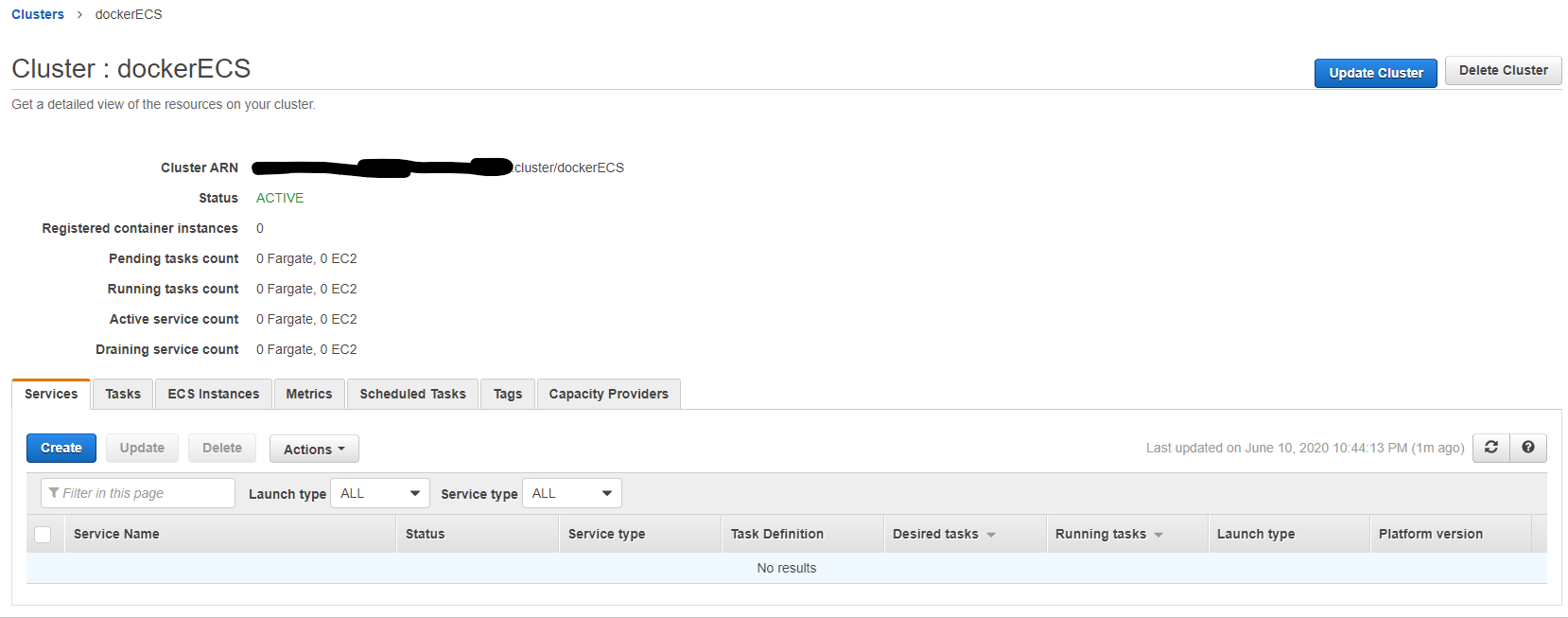
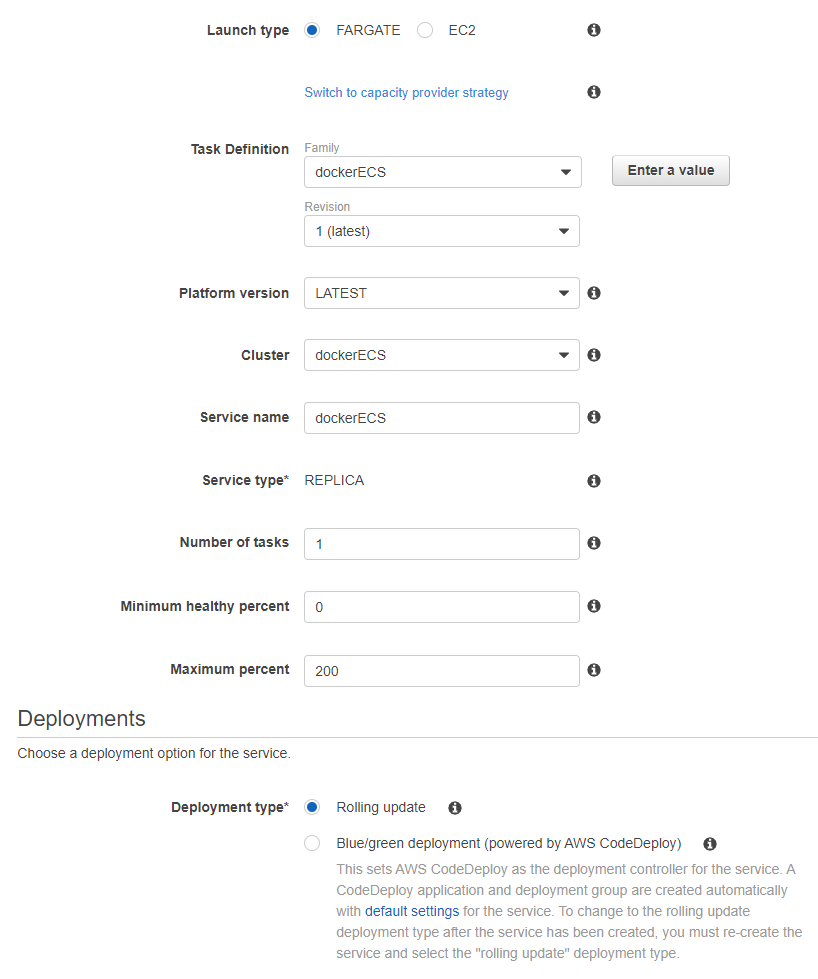
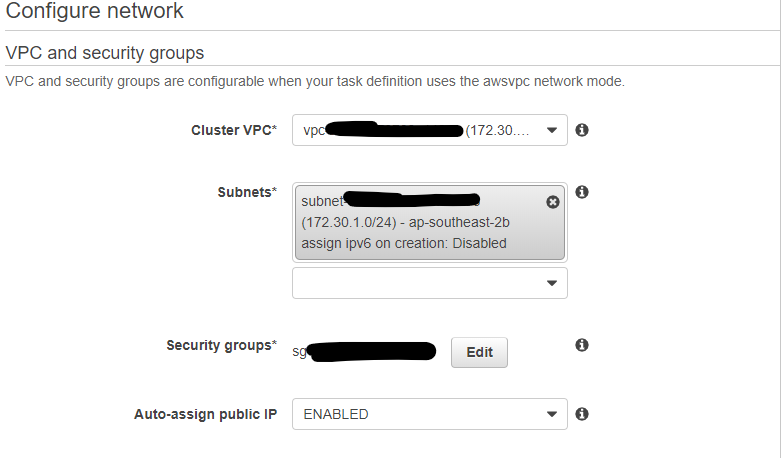
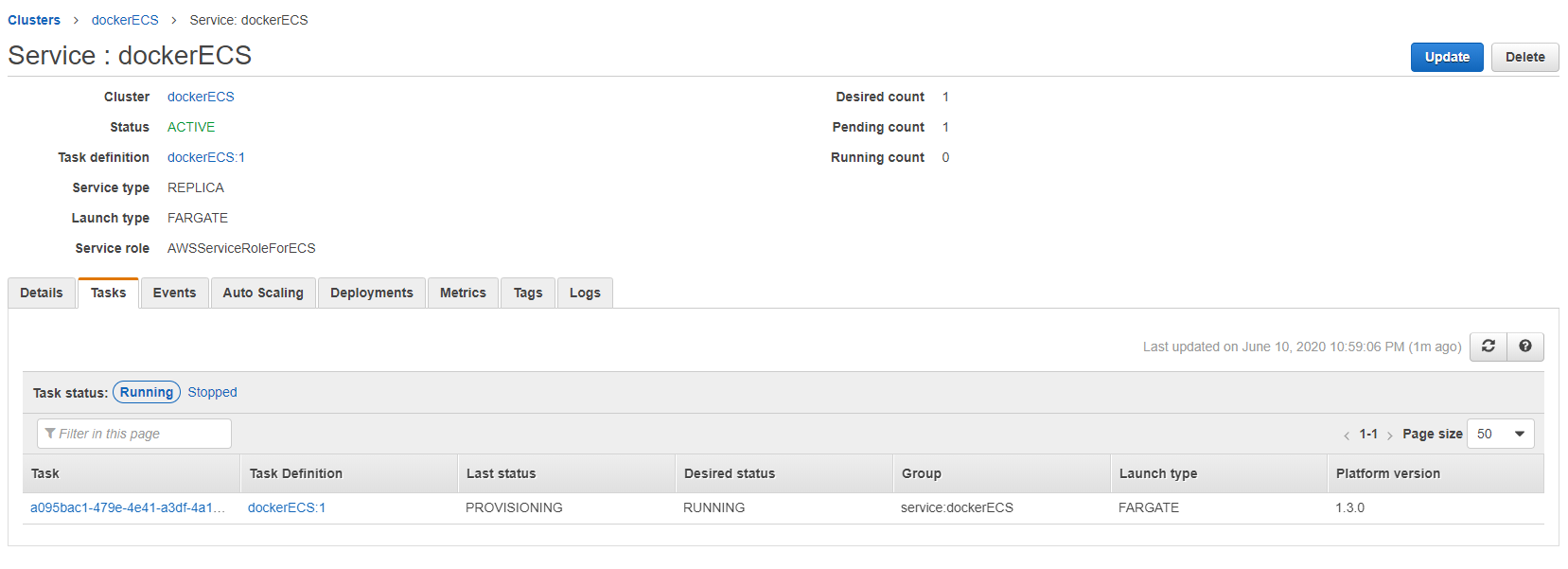
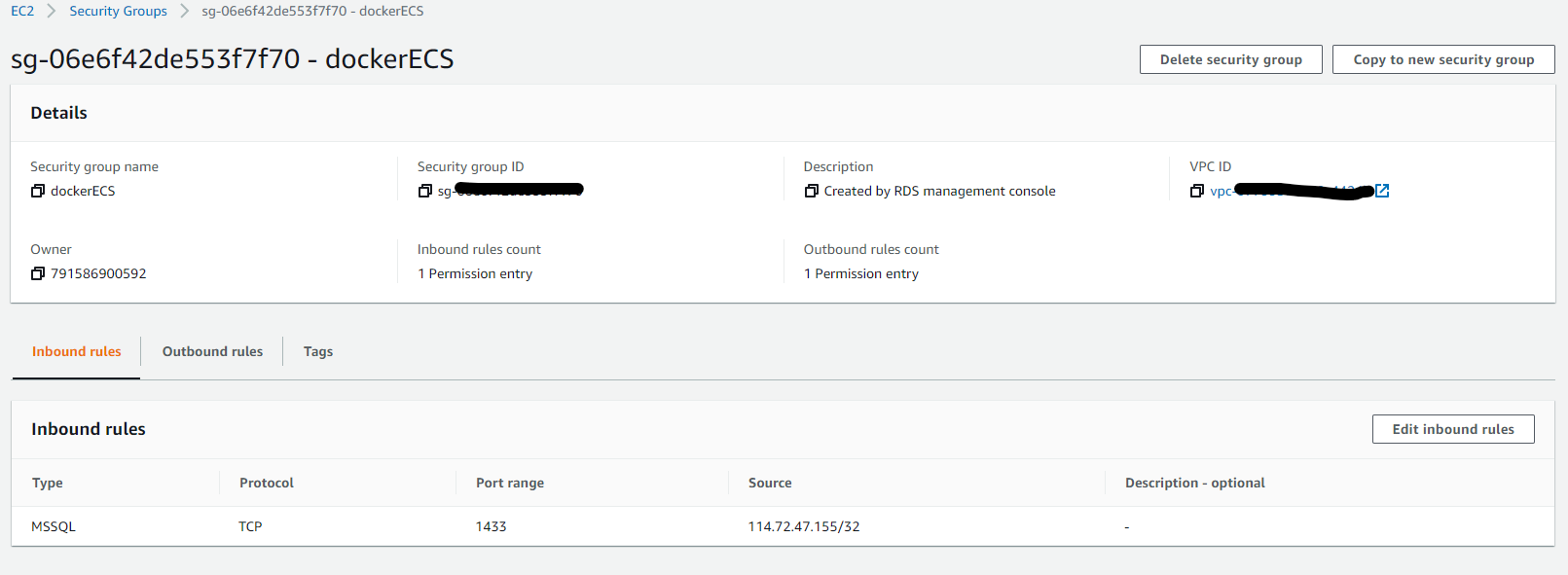
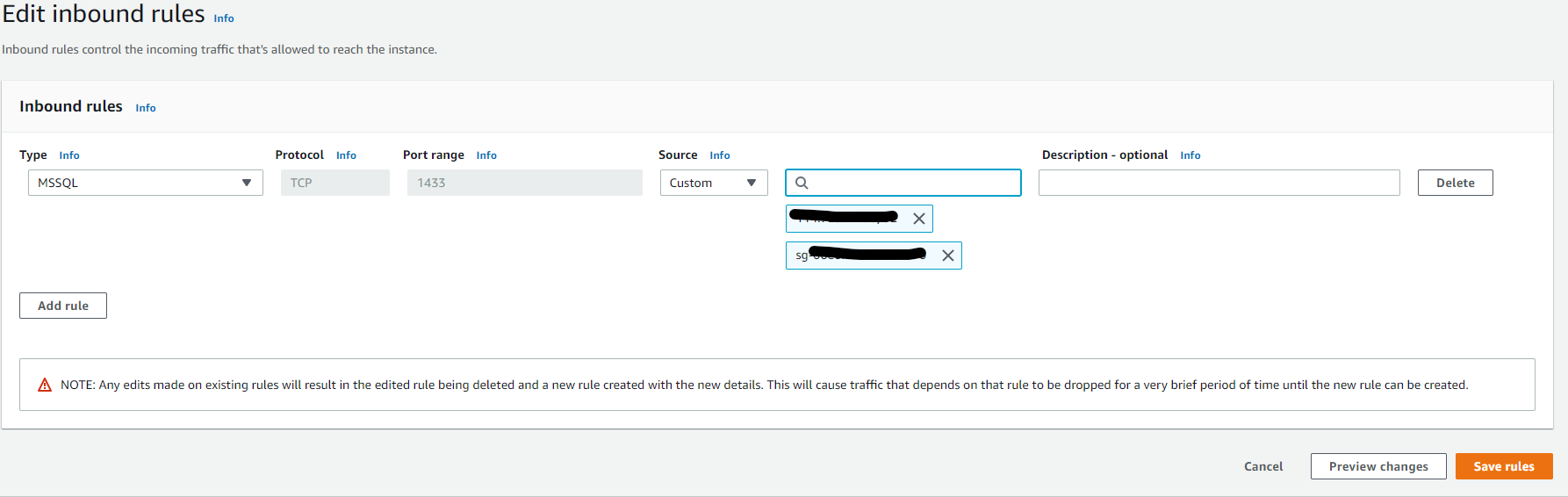
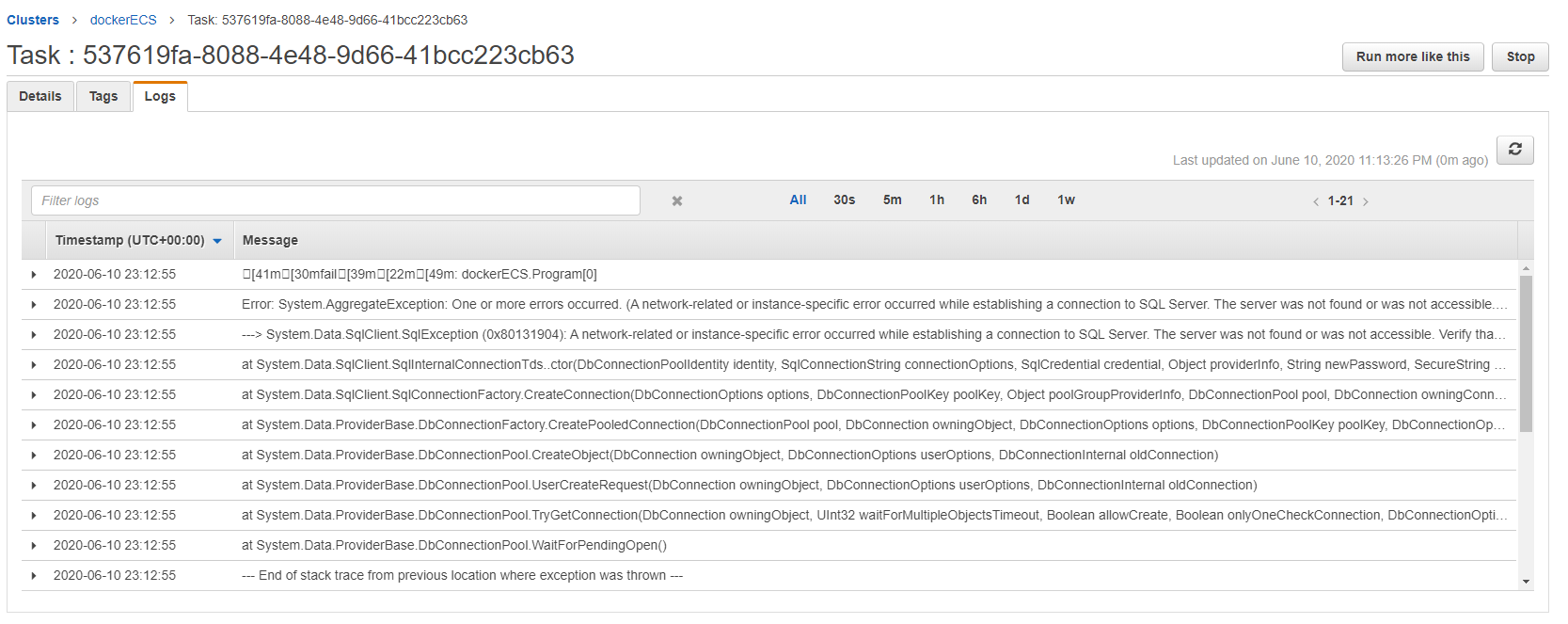
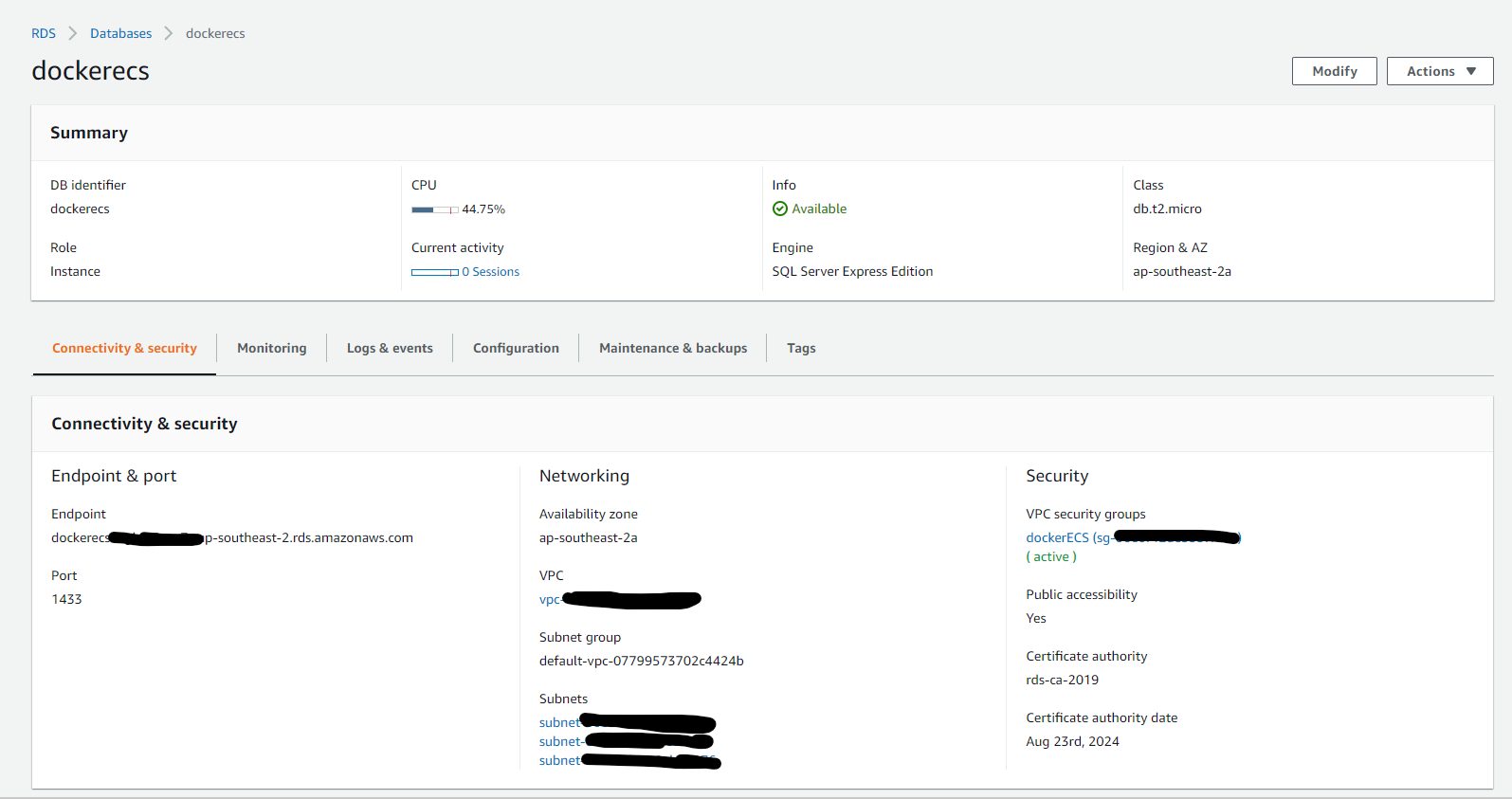
  
  
  


1. Database creation takes a while.

## 3.3 Create a RDS Database

1. Now we can create a cluster in ESC, go to AWS Console 🡪 ECS
2. We will first create a task definition, select “Task Definitions” in ECS, and click “Create new Task Definition”
3. Select “FARGATE” as launch type
4. Configure task and container with below options,  
   
5. Under “Container Definitions”, click “Add container”
6. Give a name for the container.
7. For the image URL, go to AWS Console 🡪 ECR, then select the Repository that we created earlier, copy the image URL from there and paste in container image URL input box  
   
8. Leave others as default and click “Add” at the end.  
   
9. Click “Create” to create task definition.



1. Now go to ECS 🡪 Clusters and click “Create Cluster”
2. Select Fargate for cluster template
3. Give a name for the cluster and enable CloudWatch Container Insights  
   
4. Now select the cluster and select “Create” in the Services tab  
   
5. Configure the service as below, click “Next step” at the end  
   
6. In the network configuration, select the VPC that was created during the RDS setup. You can check the value in AWS Console 🡪 RDS 🡪 select the database.
7. Select Subnet and Security groups that were created during the database creation. Leave other option as default and click “Next step”  
   
8. Leave other option as default, Create a service at the end.
9. That will create a task and start the application in a docker  
   
10. Click on the task and go to Logs tab to check the logs. Alternatively you can go to AWS Console 🡪 CloudWatch 🡪 Log groups.
11. You will notice that, out application is not running as we expected, it will throw database connectivity error.
12. Let’s fix this issue, please go to RDS and select the security group  
    
13. Click “Edit inbound rules” and add the security group that we created earlier, then Save rules.  
    
14. Check the task again in the cluster, and your application still throw the same error.  
    
15. This is because, we have uploaded the image with localhost connection string. Go ack to RDS and copy the Endpoint url  
    
16. Go back to Visual Studio and update the connection string with the endpoint URL and user credential which we created during SQL installation.  
    "DatabaseConnectionString": "Data Source=<endpoint>;MultiSubnetFailover=Yes;ApplicationIntent=ReadWrite;Initial Catalog=dockerECS;Persist Security Info=True;User ID=<user name>;Password=<password>;MultipleActiveResultSets=true;"
17. Let’s upload the image again using instruction in 3.1
18. Now the application should be able access the database successfully.  
    