DEPLOY OF A DOCKER-COMPOSE APPLICATION IN AN AWS ENVIRONMENT USING ECS CLI

Amazon ECS Containers Docker

beSharp | 23 July 2020

Nowadays, using Docker as an infrastructure container for local testing purposes is becoming more and more common. Many times developers exploit docker-compose functionality to create an infrastructure stack that contains their application, web server, and databases in different docker containers.

In this article, you will learn how to deploy an entire solution inside your AWS environment using AWS ECS Fargate service. Let's start by explaining what AWS ECS Fargate is.

Amazon ECS and AWS Fargate in a nutshell

Amazon ECS is a service that allows you to run and manage clusters of Docker containers. It's fully managed by Amazon and easily scalable based on your traffic load.

It comes in two possible flavors:

- EC2 instances hosting your containers: with this configuration, EC2 instances scaling policies are managed by the developer.
- Fargate mode: automatically manages your containers, providing the correct computational resources to your application.

AWS ECS three main actors are:

1. **Cluster**: the logical grouping of ECS resources.
2. **Service**: resource that allows you to run and maintain a specified number of instances of a task definition simultaneously, in an Amazon ECS cluster.
3. **Task-Definition**: a text file, in JSON format, that contains all the definitions and configurations of your containers.

Now that you've learned what AWS ECS Fargate is, let's try some hands-on.

Convert Docker containers to Fargate Task Definition

Before starting to create resources inside the AWS environment, you have to split your docker defined inside the docker-compose file. To do this, some considerations must be made.

Approaching Database deployment

If you have some databases inside your docker-compose, it is highly recommended to consider the adoption of the right AWS managed service, for example AWS RDS for relational databases, AWS DynamoDB for non-relational databases, or AWS ElastiCache for cache databases.

Follow a Stateless approach instead of a Stateful one.

To reach scalability, a stateless approach should be used: this means that two requests of the same user session can be executed on different instances of your container application.

Let's start creating all the AWS services you need to build your application. For AWS Fargate we have already discussed all steps needed to create a cluster and services in this article.

BUT

If you are searching for a **magical tool** that creates everything for you, you are in the right place!

Step 0: Install the ECS CLI

Recently AWS has released a new command-line tool for interacting with the AWS ECS service. It simplifies creating, updating, and monitoring clusters and tasks from a local development environment. To install it simply run this command on your CLI:

```
sudo curl -Lo /usr/local/bin/ecs-cli https://amazon-ecs-cli.s3.amazonaws.com/ecs-cli-darwin-amd64-latest
```

for downloading it inside your bin folder.

```
chmod +x /usr/local/bin/ecs-cli
```

to give executable permission.
After that, to verify that the CLI works properly, run

```
ecs-cli --version
```

Step 1: Cluster Definition

Once you have installed the CLI, you can proceed with the ECS Cluster creation. First, configure your cluster using the ECS CLI and then deploy it on your AWS account.

To configure the cluster simply run:

```
ecs-cli configure --cluster test --default-launch-type FARGATE --config-name test --re
gion eu-west-1
```

This command defines a cluster named “test” with default lunch type “FARGATE” in the Ireland region.

Now you just have to deploy it. In case your account contains a VPC that you want to use, you’ll need to specify it in the deploy command:

```
ecs-cli up --cluster-config test --vpc YOUR_VPC_ID --subnets YOUR_SUBNET_ID_1, YOUR_SU
BNET_ID_2
```

Keep in mind that if you specify a custom VPC ID you have to specify also the subnets ids where you want to deploy your service; to let ECS CLI create and configure the VPC for you, simply run:

```
ecs-cli up --cluster-config test
```

This command will create an empty ECS Cluster, and if you have not specified the VPC before, a CloudFormation stack with the VPC resources.

Another thing that we need to create is the security group for your ECS service. You can create it using the AWS CLI running these commands:

```
aws ec2 create-security-group --description test --group-name testSecurityGroup --
vpc-id YOUR_VPC_ID
```

```
aws ec2 authorize-security-group-ingress --group-id SECURITY_GROUP_ID_CREATED --proto
col tcp --port 80 --cidr 0.0.0.0/0 --region eu-west-1
```

These commands create a security group associated with the passed VPC ID and authorize ingress rules from the Internet. Take note of the ID and the security group name specified here because you will use them in the next step.

Step 2: Role Creation

Now that you have your cluster up and running, go ahead by creating the AWS IAM Role used by your Task Definition. This role contains the access policy to AWS resources for your containers. To create this Role, you first have to create a file named “assume_role_policy.json” with this content:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Then run the following command:

```
aws iam --region eu-west-1 create-role --role-name ecsTaskExecutionRole --assume-role
-policy-document file://assume-role_policy.json
```

After the role is created, run the following to attach the AWS managed policy for ECS Tasks that allow ECS containers to create AWS CloudWatch Log Group.

```
aws iam --region eu-west-1 attach-role-policy --role-name ecsTaskExecutionRole --poli
cy-arn arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
```

Step 3: Docker-Compose File and Ecs Configuration File

The next step is to modify your docker-compose file with some AWS references. Remember that, as per the current state of the art, the only supported versions of docker-compose are 1, 2, and 3.

Let's suppose that you have a docker-compose like this one:

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
```

It simply defines a web service with the NGINX image and exposes the 80 port. What you have to do now is to add the logging property as per AWS logging's best practices to manage all container logs in AWS CloudWatch, and create the ECS CLI configuration file. To add the logging property simply modify the docker-compose file as described below:

```
version: '3'
services:
  web:
    image: nginx
    ports:
      - "80:80"
    logging:
      driver: awslogs
      options:
        awslogs-group: tutorial
        awslogs-region: eu-west-1
        awslogs-stream-prefix: web
```

The logging properties contain the driver property “awslogs”, that tells ECS to log on AWS CloudWatch service. The options section defines the name of the CloudWatch log group that is automatically created from AWS, the AWS region, and the stream prefix.

Now that you have modified the docker-compose file, you have to create a new file called “ecs-params.yml” that contains the configurations of your ECS Cluster and ECS Service. In this file, you can specify:
The networking configuration with your vpc and subnets.
The Permission configuration with the role that you created in the second step.
Task configuration: properties like CPU and RAM limits for deploying the service.

For our example, let’s just define the basic configuration parameters:

```
version: 1
task_definition:
  task_execution_role: YOUR_ECS_TASK_EXECUTION_ROLE_NAME
  ecs_network_mode: awsvpc
  task_size:
    mem_limit: 0.5GB
    cpu_limit: 256
run_params:
  network_configuration:
    awsvpc_configuration:
      subnets:
        - "YOUR SUBNET ID 1"
        - "YOUR SUBNET ID 2"
      security_groups:
        - "YOUR SECURITY GROUP ID"
      assign_public_ip: ENABLED
```

In the “task_execution_role” property, just enter the name of the role that you have defined in the second step.

In the “subnets” and “security_groups” properties, enter the public subnet and the security group you’ve defined in step one.

Step 4: Deploy the docker-compose

Now that everything is configured you can deploy the solution in your AWS account through the following command:

```
ecs-cli compose --project-name test service up --create-log-groups --cluster-config te
st
```

Your application is now deployed and ready to be used!

As a bonus note: check the service status using this command:

```
ecs-cli compose --project-name test service ps --cluster-config test
```

That's all for today! In this article, we explained **how to deploy a docker-compose application inside the AWS environment with a focus on the new ECS CLI provided by Amazon**, see you soon in 14 days with the next article 🙂

#Proud2beCloud

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all’avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it