CONVERTIAMO UN'APPLICAZIONE DOCKER-COMPOSE IN AWS FARGATE SFRUTTANDO LA CLI ECS

Amazon ECS | AWS Fargate | Containers | Docker

beSharp | 23 Luglio 2020

Al giorno d'oggi, molto spesso capita di affidarsi a **Docker** per generare e gestire le infrastrutture delle applicazioni al fine di eseguire test in un ambiente locale. Molte volte, infatti, gli sviluppatori ricorrono a docker-compose per creare l'infrastruttura contenente la loro applicazione, il loro web server e i loro database in diversi container Docker.

In questo articolo, vedremo insieme come rilasciare un'intera applicazione pensata su **docker-compose** all'interno dell'ambiente AWS utilizzando il servizio **AWS ECS Fargate**. Prima di iniziare a mettere le mani in pasta, capiamo insieme cos'è AWS ECS Fargate.

Definizioni: Amazon ECS e AWS Fargate in pillole

Amazon ECS è il servizio che permette di eseguire e gestire **cluster di container Docker**. È totalmente gestito da Amazon Web Services e facilmente scalabile in base al traffico.

Le configurazioni con cui è possibile utilizzare Amazon ECS sono due:

- Istanze EC2 che contengono i container: con questa configurazione, le EC2 e la loro alta affidabilità, sono totalmente in capo all'utilizzatore del servizio.
- **Modalità Fargate**: in maniera totalmente automatica, gestisce i container fornendo le giuste risorse computazionali.

Conosciamo ora i tre attori principali:

1. **Cluster**: il raggruppamento logico delle risorse ECS.
2. **Servizio**: la risorsa che permette di eseguire e mantenere uno specifico numero di istanze, attive, delle Task Definition.

3. **Task-Definition**: un file di testo, in formato JSON, che contiene tutte le configurazioni dei container.

Ora che abbiamo capito cos'è e come funziona **AWS ECS Fargate**, entriamo nel vivo della soluzione.

Setup: Convertire i container Docker in Fargate Task Definition

Prima di iniziare a creare risorse all'interno dell'account AWS, dobbiamo dividere tutti i docker definiti all'interno del file docker-compose. Per farlo, è necessario tenere a mente alcune considerazioni:

Database definiti all'interno del docker-compose

Nel caso in cui siano presenti database all'interno del docker-compose, è altamente consigliato convertirli in risorse AWS a sé stanti mediante l'utilizzo dei giusti servizi che Amazon stessa mette a disposizione.

Per i database di tipo relazionale, ad esempio, ci si potrà servire di AWS RDS, un servizio totalmente gestito che supporta le maggiori tipologie di database relazionali quali PostgreSQL, MySQL e Oracle.

Per i database non relazionali o dedicati al cache delle applicazioni, invece, si potrà considerare l'utilizzo di AWS DynamoDB o AWS ElastiCache.

Prediligi un approccio Stateless invece di Stateful

Per permettere alle applicazioni di scalare in maniera automatica è necessario un approccio **stateless**. Questo significa che due richieste provenienti dalla stessa sessione utente, potranno essere eseguite indistintamente su differenti istanze dell'applicazione.

Ora che abbiamo chiarito tutti gli aspetti fondamentali, possiamo iniziare a creare le risorse AWS necessarie all'interno del nostro account. Di come deployare un'applicazione utilizzando AWS Fargate abbiamo parlato in questo articolo.

Se invece sei alla ricerca di un **tool magico** che crei automaticamente tutto al posto tuo, allora **sei nel posto giusto!**

Step 0: Installare la nuova ECS CLI

Recentemente AWS ha rilasciato una nuova utility da terminale per interagire con AWS ECS in grado di aiutarci a creare, modificare e monitorare cluster e task direttamente dal nostro ambiente

di sviluppo locale. Per installarla occorre semplicemente eseguire i seguenti comandi all'interno del tuo terminale:

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

per scaricarla all'interno della tua cartella bin.

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

per fornirle i permessi di esecuzione.
Fatto questo, eseguiamo

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

per verificare che tutto sia andato nel modo giusto.

Step 1: Definizione di un Cluster

Ora che abbiamo installato la CLI, possiamo iniziare a creare il nostro Cluster ECS. Per farlo dovremo prima configurarlo utilizzando la CLI ECS e poi rilasciarlo all'interno dell'account AWS.

Per configurare il cluster eseguiamo:

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

Questo comando definisce un nuovo cluster chiamato "test", che di default andrà a lanciare i nostri task in modalità "FARGATE" nella regione dell'Irlanda.

Ora, l'unica cosa che rimane da fare è rilasciarlo. In caso sul tuo account sia già presente una VPC che vuoi utilizzare con questo cluster, ti basterà specificare il VPC ID e le subnet di riferimento all'interno del comando di deploy:

```
ecs-cli up --cluster-config test --vpc YOUR_VPC_ID --subnets YOUR_SUBNET_ID_1, YOUR_SUBNET_ID_2
```

Se invece vuoi che sia la CLI ECS a creare e configurare una nuova VPC al posto tuo, puoi semplicemente eseguire:

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

Questo comando creerà un Cluster ECS vuoto e, se non hai specificato una tua VPC, uno Stack CloudFormation con all'interno le risorse della VPC.

Un'altra cosa da creare è il security group che verrà utilizzato dal nostro servizio ECS. Possiamo crearlo direttamente con la CLI AWS utilizzando questi comandi:

```
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
```

Questi comandi creano un security group associato alla VPC ID che hai passato in input al comando e ne autorizzano l'accesso sulla porta 80 da Internet. In output a questo comando vedrai l'id del security group generato. **Prendi nota** di questo id e del nome che hai specificato per crearlo, in quanto ci serviranno più avanti.

Step 2: Creazione del ruolo di esecuzione

Adesso che abbiamo il nostro cluster correttamente rilasciato e funzionante sul nostro account, dobbiamo creare un ruolo AWS IAM che verrà utilizzato dalla Task Definition. Questo ruolo contiene le policy di accesso dei nostri container alle risorse AWS. Per creare questo ruolo, creiamo prima un nuovo file chiamato "assume_role_policy.json" con questo contenuto:

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

Eseguiamo poi il seguente comando:

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

Una volta che il ruolo è stato creato, dobbiamo solo agganciare la policy che permetterà ai container di creare dei nuovi Log Group su AWS CloudWatch. Possiamo farlo semplicemente eseguendo questo comando:

```
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 e File di configurazione Ecs

Il prossimo step è quello di modificare il nostro docker-compose file con alcune configurazioni AWS. Ricordati che, al momento della scrittura di questo articolo, le uniche versioni di docker-compose supportate sono la 1, 2 e la 3.
Supponiamo di avere un docker-compose come questo:

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

Come vediamo, definisce semplicemente un servizio web NGINX, ed espone il tutto sulla porta 80. Quello che dobbiamo fare adesso è aggiungere le configurazioni per i logs come da best practices AWS. Queste configurazioni permetteranno ad AWS di gestire i log mediante il servizio AWS CloudWatch.

```
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
```

Le configurazioni per il log contengono:

1. La proprietà driver che deve essere settata con “awslogs”, la quale dice ad ECS di utilizzare il servizio CloudWatch per salvare i log del container.
2. La sezione options che definisce il nome del log group che verrà creato automaticamente su CloudWatch, la regione AWS e il prefisso per lo stream di log.

Adesso che abbiamo modificato il docker-compose, creiamo un nuovo file “ecs-params.yml” contenente tutte le configurazioni del nostro ECS Cluster e del nostro servizio ECS. In questo file potremo specificare:

- Tutte le configurazioni di networking come la vpc dove eseguire il servizio e le relative subnets.
- Il ruolo di esecuzione del nostro container, dove utilizzeremo il ruolo definito nel secondo step.
- Tutte le configurazioni per il nostro container, come CPU e RAM del container.

Per il nostro esempio, definiremo un file contenente le configurazioni di base:

```
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
```

Nel campo “task_execution_role”, andremo a inserire il nome del ruolo definito nel secondo step.

Nel campo “subnets” e nel campo “security_groups”, invece, inseriremo gli ID delle subnet pubbliche e l’id del security group definiti nel primo step.

Step 4: Deploy del docker-compose

Ci siamo! È il momento di rilasciare la nostra soluzione sull’account AWS:

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

Verifichiamo lo stato del servizio eseguendo il comando qui sotto e il gioco è fatto!

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

E anche per oggi è tutto. In questo articolo abbiamo visto insieme come rilasciare un’applicazione che fa uso di docker-compose all’interno dell’ambiente AWS, con un focus sulla nuova ECS CLI messa a disposizione da AWS.

Continuate a seguirci: appuntamento **tra 15 giorni** con il prossimo articolo 🙂

#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