

# DEPLOY DI SERVIZI SU AWS FARGATE CON PIPELINE DI CONTINUOUS DELIVERY

AWS CodeBuild

AWS CodeCommit

AWS CodePipeline

AWS Fargate

CI/CD

Continuous Delivery



beSharp | 5 Aprile 2019

---

**In questo articolo spieghiamo come abbiamo realizzato una pipeline di CD in grado di produrre una docker image ed effettuare il deploy della stessa su AWS ECS Fargate.**

Con l'avvento di **AWS Fargate** la realizzazione di servizi basati su container assume finalmente tutto un altro senso. Infatti, prima del rilascio di Fargate l'unico modo per usare Amazon ECS prevedeva il provisioning di un cluster di istanze EC2 gestite da Amazon (per software, aggiornamenti e configurazione). Con questo tipo di soluzione bisogna sostenere i costi dei cluster, progettare il sovradimensionamento per consentire lo scaling dei task, ed infine configurare e mantenere un valido sistema di autoscaling per non rimanere mai senza adeguate risorse per i container.

Con AWS Fargate tutto questo **overhead di gestione può essere lasciato ad AWS**, che ci permette di avviare servizi basati su container pagandoli solo per il tempo effettivo di esecuzione. Non c'è quindi bisogno di preoccuparsi del cluster sottostante, ed è possibile concentrarsi sullo sviluppo dei servizi.

Con AWS Fargate, AWS sta rendendo il container un oggetto di prim'ordine nel panorama delle soluzioni di computing.

Automatizzare il deploy di servizi basati su container è di fondamentale importanza per riuscire a sfruttare a pieno le potenzialità di AWS Fargate e del Cloud di AWS.

Ecco la nostra soluzione per implementare una pipeline di CD in grado di mettere in produzione ogni push sul ramo prescelto del repository.

I servizi chiave per l'infrastruttura sono

- Amazon Elastic Container Service (Amazon ECS)
- Amazon Elastic Container Registry (ECR)
- Elastic Load Balancing
- (Opzionale) CodeCommit per il repository GIT. Qualsiasi altro repository supportato da CodePipeline andrà bene.

Cominciamo con un breve glossario

**Amazon Elastic Container Service (Amazon ECS)** è un servizio di orchestrazione di container. Supporta Docker e consente di **eseguire e ridimensionare facilmente le applicazioni**. Con AWS Fargate è possibile avviare ed orchestrare servizi basati su container sfruttando cluster completamente gestiti da AWS e pagando per container.

**Amazon Elastic Container Registry (ECR)** è un **registro di immagini Docker completamente gestito** che semplifica agli sviluppatori la memorizzazione, la gestione e la distribuzione di immagini di container Docker.

Per smistare il traffico attraverso i container è possibile sfruttare il servizio di Elastic Load Balancing.

**AWS Elastic Load Balancing** instrada automaticamente il traffico in entrata delle applicazioni tra molteplici destinazioni, tra cui EC2, container, indirizzi IP e funzioni Lambda.

Elastic Load Balancing offre **tre tipi di sistemi di bilanciamento del carico**:

1. Application Load Balancer
2. Network Load Balancer
3. Classic Load Balancer

Il load balancer che ci sarà utile per i servizi rilasciati su Fargate è **l'Application Load Balancer (ALB)**

I sistemi Application Load Balancer sono indicati per il **bilanciamento di traffico HTTP e HTTPS**, e offrono instradamento avanzato delle richieste per la distribuzione di architetture moderne, ad esempio in microservizi e container. Questi sistemi operano a livello di richieste individuali (livello 7) e instradano il traffico in base al contenuto della richiesta.

Senza ulteriori indugi, passiamo ora al tutorial per la realizzazione di una pipeline di rilascio completamente automatizzata.

Durante il resto dell'articolo daremo per scontato che tutto il codice del progetto si trovi su un **repository compatibile con CodePipeline**.

## Parte 1: Preparazione della prima docker image

La prima cosa da fare è preparare un'immagine del nostro servizio per poterla testare sia in locale che su AWS.

Occorre quindi **aggiungere al progetto un Dockerfile**, che andrà successivamente pubblicato sul repository. All'interno del file dovranno essere indicate le istruzioni per costruire un container che contenga tutto il software, le dipendenze, le librerie, le configurazioni ed il pacchetto con il nostro servizio.

Questo container può essere tranquillamente testato in locale o in un ambiente controllato per verificarne il corretto funzionamento.

Una volta soddisfatti del risultato dei test locali, si può procedere alla creazione di un'immagine e alla pubblicazione della stessa su Amazon ECR.

Provvediamo quindi alla **creazione di un repository ECR**: l'unico dato necessario alla creazione è un nome valido.

Successivamente basta seguire le istruzioni di login e push per caricare la nostra immagine docker su ECR.

```
$(aws ecr get-login --no-include-email --region <regione>)  
docker build -t <nome immagine> .  
docker tag <nome immagine>:latest <ecr url>:latest  
docker push <ecr url>:latest
```

## Parte 2: Configurazione di ECS Fargate e del Networking

Il nostro servizio avrà bisogno di un **Load Balancer** per instradare il traffico tra i container replica.

Per questa ragione dobbiamo **creare un Application Load Balancer**, la cui configurazione può essere lasciata in bianco. Non sarà necessario definire dettagli di comportamento dell'ALB perchè sarà ECS a gestirlo in modo dinamico durante le operazioni di scaling dei container.

Per quanto riguarda ECS, la prima cosa da fare è creare un cluster. **I cluster** non sono altro che oggetti utilizzati per raggruppare a livello logico i servizi.

## Select cluster template

The following cluster templates are available to simplify cluster creation. Additional configuration and integrations can be added later.

**Networking only**  
Resources to be created:  
Cluster  
VPC (optional)  
Subnets (optional)  
**Powered by AWS Fargate**

**EC2 Linux + Networking**  
Resources to be created:  
Cluster  
VPC  
Subnets  
Auto Scaling group with Linux AMI

**EC2 Windows + Networking**  
Resources to be created:  
Cluster  
VPC  
Subnets  
Auto Scaling group with Windows AMI

\*Required Cancel **Next step**

Per creare un cluster accediamo alla dashboard di ECS e selezioniamo “create cluster”.

Dal wizard scegliamo “Networking only”, configurazione che indica ad AWS di utilizzare AWS Fargate per questo cluster virtuale.

Nel secondo ed ultimo step del wizard basta indicare il nome e se si desidera creare una nuova VPC, se si sceglie di non creare nessuna VPC è possibile usare una di quelle già configurate sul proprio account.

Configure cluster

Cluster name\*  ⓘ

**Networking**  
Create a new VPC for your cluster to use. A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Fargate tasks.

Create VPC  Create a new VPC for this cluster

**Tags**

Key	Value
<input type="text" value="Add key"/>	<input type="text" value="Add value"/>

\*Required Cancel Previous **Create**

Il secondo passaggio è **creare una task definition**. Questo oggetto raccoglie informazioni sul task come ad esempio nome, descrizione, IAM roles per deploy ed esecuzione, la dimensione del task in termini di RAM e CPU, e le specifiche del container che lo ospiterà.

Per la parte di configurazione del container occorre selezionare l’immagine docker precedentemente salvata su ECR.


Per creare una task definition basta selezionare “Create task definition” dall’apposita schermata dall’area di ECS.

Il passaggio essenziale è scegliere AWS Fargate al primo step del wizard; completiamo poi con i dati richiesti seguendo le istruzioni e fornendo dimensionamento adeguato per il proprio task.

## Select launch type compatibility


Select which launch type you want your task definition to be compatible with based on where you want to launch your task.

**FARGATE**



Price based on task size  
Requires network mode awsvpc  
AWS-managed infrastructure, no Amazon EC2 instances to manage

**EC2**



Price based on resource usage  
Multiple network modes available  
Self-managed infrastructure using Amazon EC2 instances

\*Required

Cancel

Next step

## L'ultimo oggetto da configurare si chiama servizio (Service).

Un servizio è definito da un task e da una terna di parametri che specificano quante istanze del task sono necessarie come minimo, valore attuale e massimo per permettere il corretto funzionamento del servizio.

Cluster : cd-test Delete Cluster

Get a detailed view of the resources on your cluster.

Status: **ACTIVE**

Registered container instances: 0

Pending tasks count: 0 Fargate, 0 EC2

Running tasks count: 0 Fargate, 0 EC2

Active service count: 1 Fargate, 0 EC2

Draining service count: 0 Fargate, 0 EC2

Services | Tasks | ECS Instances | Metrics | Scheduled Tasks | Tags

Create | Update | Delete | Actions

Filter in this page | Launch type: ALL | Service type: ALL | Last updated on March 15, 2019 11:09:08 AM (1m ago) | 1/4

Service Name	Status	Service type	Task Definition	Desired tasks	Running tasks	Launch type	Platform version
<a href="#">HelloWorld</a>	ACTIVE	REPLICA	blog-cd-test2	0	0	FARGATE	LATEST(1.3.0)

La procedura di creazione è analoga a quella degli altri oggetti configurati.

## Configure service

A service lets you specify how many copies of your task definition to run and maintain in a cluster. You can optionally use an Elastic Load Balancing load balancer to distribute incoming traffic to containers in your service. Amazon ECS maintains that number of tasks and coordinates task scheduling with the load balancer. You can also optionally use Service Auto Scaling to adjust the number of tasks in your service.

Launch type  FARGATE  EC2 ⓘ

Task Definition Family:

Revision:

Cluster:  ⓘ

Service name:  ⓘ

Service type\*  REPLICA  DAEMON ⓘ

Number of tasks:  ⓘ

Minimum healthy percent:  ⓘ

Maximum percent:  ⓘ

### VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC\* vpc-a2918ac0 (10.101.0.0/16) | be... ⓘ

Subnets\* ⓘ

- subnet-839ca8c5 (10.101.2.0/24) | beSharp-dev-public-c - eu-west-1c assign ipv6 on creation: Disabled
- subnet-af739dca (10.101.0.0/24) | beSharp-dev-public-a - eu-west-1a assign ipv6 on creation: Disabled
- subnet-182b3c6c (10.101.1.0/24) | beSharp-dev-public-b - eu-west-1b assign ipv6 on creation: Disabled

Security groups\* HelloW-4568 Edit ⓘ

Auto-assign public IP ENABLED ⓘ

### Load balancing

An Elastic Load Balancing load balancer distributes incoming traffic across the tasks running in your service. Choose an existing load balancer, or create a new one in the [Amazon EC2 console](#).

Load balancer type\*  None  
Your service will not use a load balancer.

Application Load Balancer  
Allows containers to use dynamic host port mapping (multiple tasks allowed per container instance). Multiple services can use the same listener port on a single load balancer with rule-based routing and paths.

Network Load Balancer  
A Network Load Balancer functions at the fourth layer of the Open Systems Interconnection (OSI) model. After the load balancer receives a request, it selects a target from the target group for the default rule using a flow hash routing algorithm.

Classic Load Balancer  
Requires static host port mappings (only one task allowed per container instance); rule-based routing and paths are not supported.

Service IAM role Task definitions that use the awsvpc network mode use the AWSServiceRoleForECS service-linked role, which is created for you automatically. [Learn more](#).

Load balancer name Helloworld-alb ↕

### Container to load balance

Container name : port HelloWorldContainer:80:80 Add to load balancer

Si faccia attenzione alla scelta della VPC, delle subnet e del load balancer creato precedentemente.

Alla fine della configurazione, puntando il browser all'URL dell'ALB si dovrebbe visualizzare il proprio servizio.

# Hello world

## Hello from PHP7!

---

# A random number

8442

## Parte 3: Automatizzare il deploy

Una volta configurato manualmente tutto l'ambiente, è possibile **creare e configurare una pipeline per effettuare il deploy automatico ad ogni cambiamento del codice.**

Prima di iniziare la configurazione della pipeline occorrerà aggiungere un file chiamato **buildspec.yml** nella root del repository. Lo scopo del file è quello di contenere le istruzioni per effettuare il build di una nuova immagine del nostro servizio.

Dobbiamo infatti automatizzare il processo che abbiamo eseguito a mano nella parte iniziale dell'articolo, ovvero la costruzione dell'immagine docker a partire dal dockerfile e dal codice, il suo caricamento su ECR ed infine l'aggiornamento di ECS per effettuare il deploy del servizio usando la nuova immagine.

Ecco una versione di esempio del file da aggiungere (buildspec.yml)

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - $(aws ecr get-login --region $AWS_DEFAULT_REGION --no-include-email)
      - REPOSITORY_URI=<URL DELL'IMMAGINE SU ECR>
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name": "<container name usato in task definition>", "imageUri": "%s"}]' $REPOSITORY_URI:$IMAGE_TAG > imagedefinitions.json
artifacts:
  files: imagedefinitions.json
```

*In rosso le parti da editare con i nomi specifici del proprio progetto.*

CodePipeline è il servizio di AWS che ci permetterà di realizzare l'automazione profondendo bassissimo effort di gestione e configurazione. Per eseguire le operazioni di build dell'immagine invece, ci affideremo a CodeBuild, che andrà ad eseguire il file con le istruzioni (buildspec.yml).

Cominciamo quindi con la **configurazione della pipeline** creandone una nuova su CodePipeline:

1. Selezionare "Create pipeline"

2. Seguiamo il wizard, e senza alcun effort aggiuntivo, sarà possibile aggiornare il proprio servizio in modo automatico
3. Nel secondo passaggio, come sorgente occorre scegliere il proprio repository GIT (CodeCommit, o compatibile).
4. Il passaggio successivo serve a configurare il passo di Build: occorre selezionare CodeBuild e l'opzione per crearne uno nuovo
5. Nel sotto-wizard bisogna dare un nome univoco al progetto di build. Come sistema operativo bisogna scegliere **Ubuntu**, e come Runtime **Docker**.
6. Selezionare quindi la versione di docker più adatta ai propri scopi.
7. Salvare completando l'ultimo step del sotto-wizard e tornare ad editare la pipeline.
8. Si configura quindi il passo di Deploy, per il quale è possibile scegliere **Amazon ECS**
9. Allo step 4 bisogna indicare il nome del cluster precedentemente configurato a mano e il nome del servizio da aggiornare
10. Per lo step 5 basta selezionare l'opzione per permettere a CodePipeline di creare un ruolo per l'esecuzione della pipeline e di configurarlo al posto nostro. Questo ruolo viene usato da CodePipeline per modificare ECS, è gestito da Amazon ed ha il set minimo di permessi per far funzionare tutto.
11. Salvare la pipeline.

A questo punto, la pipeline proverà ad andare in esecuzione in automatico, *fallendo*.

**Si tratta di un fallimento atteso:** la ragione risiede nel fatto che il Wizard ha creato per noi un ruolo per CodeBuild, che però non ha tutti i permessi necessari per eseguire il push dell'immagine su ECR.

Per porre rimedio occorre identificare il ruolo generato – il cui nome segue la seguente convenzione: code-build-**build-project-name**-service-role – e aggiungervi i seguenti permessi:

**AmazonEC2ContainerRegistryPowerUser** per vedere funzionare la pipeline.

Se tutto ha funzionato come previsto, la pipeline sarà ora funzionante e, ad ogni commit sul ramo prescelto, il servizio sarà automaticamente aggiornato.

Con i container sempre più al centro della scena DevOps è importante conoscere gli strumenti a disposizione per effettuare scelte di progetto ponderate ed efficaci. Speriamo di esservi stati utili in questo senso 😊

Condividete con noi i vostri risultati, le vostre osservazioni, i vostri dubbi, i vostri spunti... il nostro team non vede l'ora di approfondire il topic con voi!

[ATTENZIONE, SPOILER!]: se siete incuriositi da questo argomento, continuate a seguirci: in arrivo per voi **un modo creativo per ottenere una pipeline automatica, altamente personalizzata, che utilizza i container come mezzo di automazione.**

Stay tuned!



**#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

Copyright © 2011-2021 by beSharp srl - P.IVA IT02415160189