ECS -> orchestratore minimale di container, lightweight rispetto a Kubernetes originale  
EKS -> distro di Amazon di Kubernetes, orchestratore a tutti gli effetti

S3-> servizio per storage di file in cloud Amazon  
Cognito -> IAM di Amazon (usato da OpenSearch per esempio)  
OpenSearch su AWS -> distro opensource di Elastic, basata su Lucene  
EC2 -> servizio amazon per creare VM  
Fargate-> servizio amazon per configurare VM serverless  
  
ECS si puo’ usare con EC2 o con Fargate  
  
ECR-> il registro immagini docker di Amazon  
RDB -> servizio AWS per creare database relazionali  
Parameter Store -> servizio in cloud AWS per salvare le mappe di parametri  
Secret Manager -> servizio in cloud AWS per salvare i secrets  
  
Se configuri le dipendenze di spring boot per amazon cloud, puoi interagire con alcuni dei servizi di cui sopra.  
Ad esempio se le metti come dipendenze maven, e poi crei un file bootstrap.properties, quando l’applicativo spring boot sale cerca nel bootstrap properties per esempio la convenzione dei file di configurazione da iniettare nell’environment (un po’ come avviene nei deployment file per kubernetes).  
La convenzione delle properties quindi nel parameter store per AWS è :  
/prefix/appname/nome.del.parametro

Appname è il nome che si configura nell ‘application.properties del progetto spring es  
spring.application.name=letitout  
quindi per esempio se la nostr applicazione è definita con name letitout e sul Parameter Store di AWS

Mettiamo  
/config/letitout/userdatabase.hostname  
  
allora nel mio application properties potro’ avere nella stringa dell’url del db per esempio

spring.datasource.url=jdbc:mariadb://${**userdatabase.hostname**}:3306/dbname  
visto che viene fatto l’injection di userdatabase.hostname che si prende dal parameter store AWS sul cloud.  
  
NB: per altri servizi è necessario fornire parametri specifici nell’application.properties per AWS, ad esempio tipicamente la coppia di key/name per l’authorization (access key name e value)

Su AWS se si crea un **task definition + un container** (le specs del container) associato, se vengono deployate in un **cluster** (creato sempre su AWS) diventano un **Servizio** AWS

# Come creare pipeline github > ecs (su ec2) spring boot, che si collega a db (RDB), AWS S3

**PRIMO DEPLOY SPRING BOOT**

**Creare l’app spring boot.  
CREARLO PER ORA SENZA CONNETTORI AL DB, AD S3 o l’USO DI PARAMETRI DA INIETTARE DAL ParameterStore. Settare il server.port su 8080 nelle properties. (aggiungeremo i pezzi man mano)**  
Nell’esempio usiamo jdk 17  
Configurare nel pom il plugin Buildpack che permette di buildare l’immagine docker senza dover separatamente fare la build con maven e poi usare il docker file (il plugin è già incluso in spring boot)  
  
**Installare AWS Tools cli.**  
  
**Push della docker image dell’applicazione sul repository AWS (ECR)**Fare la build mvn e creazione della docker image (con mvn spring-boot:build-image -Dspring-boot.build-image.imageName etc) questo metterà l’immagine docker nel registro locale del docker.  
Andare sul servizio ECR dalla console di AWS ,scegliere (Repositories).  
Fare create repository.  
Una volta creato il repository, andare in alto a destra su **view push commands** e seguire le indicazioni per , tramite la AWS cli, per pushare dal repository docker locale a quello di AWS.  
  
**Creazione del task definition**Andare dalla console di AWS sul servizio **ECS**.  
Scegliere la vecchia visualizzazione  
A screenshot of a computer

Description automatically generated  
Andare su task definitions, fare **Create new task definition**Scegliere EC2  
A screenshot of a computer

Description automatically generated

Inserire il task definition name, e settare solo il taskExecutionRole (con create new o ad uno esistente) ma lasciare libero il task Role !  
A screenshot of a computer

Description automatically generated  
  
lasciare task memory e cpu vuoti e cliccare **Add Container**  
A screenshot of a computer

Description automatically generatedInserire il nome del container associato al task, come image il link dell’immagine sul repository di AWS (ECR) e inserire un Memory **HARD LIMIT** che deve essere < del totale di RAM del tipo di macchina che sceglieremo per EC2 piu’ avanti per il cluster.  
Nell’esempio stiamo usando una macchina t3.small che ha 1 GB di ram per il nostro EC2 su cui deployare, quindi 900 va bene come hard limit.  
In port mappings mappare una porta (sarà quella che chiameremo) a quella dell’applicativo che deployeremo (spring boot, di default lo mettiamo sull’8080)  
  
A screenshot of a computer

Description automatically generated

In storage e logging abilitare l’autoconfiguration del logging

A screenshot of a computer

Description automatically generated

Cliccare su **ADD e** infinite su **CREATE**.  
  
Ora abbiamo creato un task definition e un container associato.  
Se vogliamo modificare un task definition/container associato, si crea una nuova revision di quel task. E’ quello che in realtà accade quando viene pushata una nuova versione dell’immagine di un’applicativo in una action GitHub -> la nuova versione del repository finisce in una nuova revision del task definition (tutto automatizzato ovviamente)  
  
Ora Dobbiamo creare un cluster, e fare un primo deploy a mano di questo task definition + container.  
Un task definition (e container) deployati in un cluster diventano un servizio.  
A quel punto potremo automatizzare il tutto usando le action di GitHub per collegare la repo del codice GitHub direttamente al task definition e al deploy sul cluster.  
Quindi ora torniamo a creare il cluster.  
  
Andiamo su **cluster** e clicchiamo su **create cluster**.  
Sempre con la visualizzazione vecchia selezioniamo EC2+Linux Networking  
  
A screenshot of a computer

Description automatically generated

Inseriamo il nome e selezioniamo come EC2 instance type un t3.small

A screenshot of a computer

Description automatically generated  
  
Selezioniamo la vpc esistente (o creiamone una nuova se non già esistente)

A screenshot of a computer

Description automatically generated

Creiamo un nuovo security group e settiamo 8080 come port range (inbound rule degli ip che possono accedere)  
  
A screenshot of a computer

Description automatically generated  
  
A screenshot of a computer

Description automatically generated  
  
Clicchiamo su **create**.  
  
Ora creiamo il service **(cioè l’effettivo deploy del task definition sul cluster).  
Passiamo alla visualizzazione nuova  
A screenshot of a computer

Description automatically generated  
  
Andiamo su cluster sempre, selezioniamo il cluster e facciamo create service**.  
Launch type scegliamo EC2, service come tipologia  
Come family scegliamo il nome del task definition selezionato.  
  
A screenshot of a computer

Description automatically generated  
  
clicchiamo **Create  
  
Se torniamo in cluster, e services, possiamo cliccare sul service che sta venendo deployato.  
A screenshot of a computer

Description automatically generated  
  
Se a quel punto andiamo su deployment possiamo vedere lo stato del deploy  
  
A screenshot of a computer

Description automatically generated**NB: quella fatta fino ad ora è una configurazione **vanilla,** nel senso che permette di deployare un applicativo spring boot stand alone (cioè che non dipende da database, s3 o parameter store).  
  
A questo punto il nostro applicativo è deployato.  
Per vedere a quale ip risponde occorre andare sul servizio **EC2 > Instances , cliccare sull’instanza che fa parte del cluster su cui abbiamo deployato e andare su networking**A screenshot of a computer

Description automatically generated

**Se proviamo a contattarlo pero’ potrebbe comunque non andare. Questo perché bisogna modificare l’inbound rule associata al security group del cluster.  
Il security group del cluster (non confondere con l’IAM policy) rappresenta il set di regole di quali ip possono contattare in entrata o uscita.  
Per modificare quindi e abilitare tutti (che non è settato di default) andare sempre su EC2 > instance, cliccare sull’istanza, andare su security e cliccare sul SECURITY GROUP associato.  
Fare edit inbound rule e aggiungere 2 inbound rules, tutte di type All Traffic, per tutti gli ipV4 e V6  
  
A screenshot of a computer

Description automatically generated**NB: per stoppare un’istanza, andare in EC2 > autoscaling, selezionare l’autoscaling associato all’istanza EC2 (l’autoscaling è AWS che se un’istanza viene stoppata la riavvia) e settare il desidered a 0, e poi andare in instances e stoppare. In questo modo AWS quando la stoppiamo non la riavvia.

**-------------------------------------------------------------------------------------------------a**  
  
**USO DI PARAMETER STORE**

**nb: Quando abbiamo creato il task definition, abbiamo scelto un taskExecutionRole creato da AWS. Il role creato è un ruolo (che se non esiste già crea AWS) con cui gira un task di deploy che ha i permessi necessari per prendere l’immagine da ECR e deployarla. Per i task definition generalmente non c’e’ bisogno di aggiungere permessi ulteriori a quelli di default associati da AWS in creazione visto che tutto quello che deve fare un taskExecutionRole è deployare.**Un ruolo ha associate una o piu’ policy. Ciascuna policy ha uno o piu’ permessi.  
Per vedere le policies e i permessi andare sul servizio **IAM, selezionare ROLES, e cliccare sul role che si vuole visionare**A screenshot of a computer

Description automatically generated  
  
**Allo stesso modo quando abbiamo definito il cluster abbiamo scelto un container instance IAM role, fatto creare di default da AWS.  
Questo è un ruolo piu’ importante, al quale potrebbe essere necessario aggiungere policy non presenti di default, in quanto questo è il ruolo associato alla nostra istanza EC2.  
Quindi ad esempio se la nostra istanza, deve potersi collegare al Parameter Store, dobbiamo aggiungere una policy aggiuntiva.  
Per aggiungere policy si va sempre sulla console IAM, su roles, si fa AddPolicy e si sceglie la policy (o si fa l’add di un singolo permesso se non si vuole aggiungere tutta la policy). Se si vuole vedere se un ruolo permette di fare o meno determinate azioni, in base alle policy che gli abbiamo aggiunti, si puo’ fare simulate.  
  
Nel caso specifico , se configuriamo il nostro applicativo SpringBoot per collegarsi al Parameter Store , dobbiamo garantire all’istanza EC2 su cui viene deployata (il cluster) che ci sia il permesso per fare la GetParameterFullPath . Per fare questo quindi dobbiamo aggiungere la policy AdministratorAccess al ruolo che abbiamo associato all’istanza ec2 (nel nostro esempio ecsInstanceRole)**

**A screenshot of a computer

Description automatically generated**

**Quindi ora possiamo che abbiamo abilitato l’istanza EC2 a usarlo, possiamo usare ParameterStore.**

Parameter Store permette di centralizzare i parametri da usare, come le configMap di kubernetes.  
Per usarlo , innanzitutto vanno aggiunte le dipendenze da AWS nel pom.xml dell’applicazione.   
  
**A screen shot of a computer program

Description automatically generated**  
  
Quando le trova, e starta, l’applicativo cerca un file bootstrap.properties (nel classpath) in cui sono definite le convenzioni dei parametri sul parameter store  
  
**A screen shot of a computer program

Description automatically generated**  
Quindi se andiamo sul servizio AWS su **Parameter Store** possiamo aggiungerli da lì  
  
**A screenshot of a computer

Description automatically generated**  
  
e referenziarli nei properties classici di SpringBoot **(RICORDA DI DARE UN APPLICATION NAME ALL’APPLICATIVO SPRING BOOT CHE MATCHI QUELLO DOPO IL PREFIX NEI NOMI DEI PARAMETRI SULLO STORE)  
  
A screen shot of a computer code

Description automatically generated**

**Quindi una volta aggiunte le dipenze al pom.xml, il bootstrap.properties a iniettate le properties si ribuilda,e aggiunti i parametri sul Parameter Store dobbiamo rideployare la nuova versione che li usa.  
-> si ribuilda in locale, si rifa’ l’immagine e la si ripusha sul repository ECR.  
-> Si cancella quindi il servizio dal cluster , si va sul task definition precedentemente deployato, SI CREA UNA NUOVA REVISION, che prenderà la nuova versione dell’immagine, e si rideploya sul cluster come servizio.**

**-----------------------------------------------------------------------------------------------------------------------------**

CREAZIONE DATABASE RDB

**Per creare un database, andare su RDB servizio AWS, fare create database, scegliere il db di preferenza.  
Selezionare username e pw, freetier come tipologia.  
Una volta creato selezionarlo, andare in security > Security Group e definire le Inbound Rules come fatto per l’istanza EC2, per permettere le connessioni da tutti gli IpV4 e V6.  
A quel punto ci si puo’ collegare all’IP del database (visionabile dal dettaglio) e creare il primo DB.  
Si possono quindi creare i connettori nell’applicativo SpringBoot come solito, rideployare e ripushare , creando la nuova revision del task definition con la nuova versione dell’immagine sull’ECR (giro solito) , riportare nel cluster e si avra’ l’applicativo collegato al DB.**

**----------------------------------------------------------------------------------------------------------------------------**

CONFIGURAZIONE S3 PER COLLEGARE APPLICATIVO AI BUCKET  
S3 Buckets di AWS permette di avere uno storage da usare come documentale sul quale uppare file da applicativo SpringBoot.  
Per usarlo andare sul servizio AWS S3 Bucket, fare Create bucket.  
**A screenshot of a computer

Description automatically generated  
   
A screenshot of a computer

Description automatically generated  
  
Fornire Bucket Name e region (associata all’account AWS)  
Fare create bucket.  
  
Il Bucket S3 è pronto, per poterlo usare il nostro account ha bisogno di una coppia di chiavi security credentials.  
  
Andare sul profilo in alto a destra, my security credential > security credentials.  
Generare Access KeyID and Secret Access Key cliccando su CreateAccess Key  
  
Le dipendenze al pom di AWS le abbiamo gia’ aggiunte per il Parameter Store (altrimenti aggiungerle).  
Aggiungere nei properties i rifermenti all’access key e secret ottenuti sopra  
  
A computer code with text

Description automatically generated with medium confidence  
(questi si possono tenere nascosti anche nel parameter store e semplicemente iniettati qui)  
  
A questo punto ci serve di configurare a livello programmatico l’uso di S3  
  
A screenshot of a computer

Description automatically generated  
   
  
e potremo a livello di codice fare upload/download di file  
  
A screenshot of a computer program

Description automatically generated**

Quindi per vedere le modifiche cancellare il service che gira, ribuildare l’applicativo spring boot, rigenerare immagine , ripushare sul repository ECR, creare una nuova revision del task definition e rideployare sul cluster come service.

-----------------------------------------------------------------------------------------------------------------

Ora tutto questo l’abbiamo fatto deployando manualmente.  
Vedremo come collegare il repository GitHub con le GitHub Actions al deploy su AWS, in maniera tale che quando si pusha sul branch, automaticamente viene fatto il deploy.  
  
Per agganciare github e aws occorrono 3 cose:  
1)utenza AWS da associare a github, che gli permetta di lavorare con l’ECR (su cui pushare l’immagine docker che genera)  
2)file task-definition.json , da mettere nel progetto, che indichi su quale task deifnition aggiornare la revision che poi verrà buildata .  
3)un file pipeline.yml da mettere in una cartella .github/workflows che definisce **la pipeline (action) di GitHub.**  
Quindi per il file task-defintion:  
un task-definition.json che è un file che va creato a livello del pom.xml, che deve contenere il json del task definition deployato per la prima volta a mano.  
  
Quindi andiamo su ECS > task definition >json, e copiamo il json nel task-definition.json  
  
A screenshot of a computer program

Description automatically generated  
  
A screen shot of a computer

Description automatically generated  
  
questo dice su quale task definition aggiornare la revision quando l’action di github avvia il flusso.  
  
A questo punto dobbiamo creare un ruole per GitHub in AWS.  
Andiamo in servizio **IAM > Users > Create user.**Definire username, fare Attach Policies directly e dargli 3 Policies:  
A screenshot of a computer

Description automatically generated  
  
fare create user.  
Quando è creato, selezionare l’user creato , andare su **Security Credentials,** fare **Create Access Key**, e creare un access key  
  
A screenshot of a computer

Description automatically generated  
  
salvandosi keyname e secret.  
  
A questo punto abbiamo creato un user in AWS da far usare a Github. Quindi dobbiamo metterlo su github.  
Andiamo sull’account github, in settings, secrets and variables > actions  
  
A screenshot of a computer

Description automatically generated   
  
e aggiungere 2 repository secret, uno in cui mettiamo il name della chiave generata sopra, e una il secret (NB: il property name , nell’esempio AWS\_ACCESS\_KEY\_ID e AWS\_SECRET\_ACCESS\_KEY li scegliamo noi, e sono il modo in cui ci riferiremo a queste variabili nel pipeline.xml)  
  
A screenshot of a computer

Description automatically generated  
  
  
Ora rimane da creare il file .github/workflows/pipeline.xml (a partire dalla root col pom.xml) in cui mettere la configurazione della pipeline (action) di GitHub, cioè la sequenza di operazioni da effettuare.  
Le operazioni nel pipeline.xml rappresentano la build dell’applicativo, la generazione dell’immagine docker (nel nostro caso usando con mvn il BuildPack facciamo tutto assieme) poi il push sul repository ECR, e l’avvio del deploy. In questo pipeline.xml si usano le credenziali dell’utenza github creata su AWS prima e messa sulle GitHub action.  
  
A screenshot of a computer program

Description automatically generated  
  
il contenuto del file è abbastanza autoesplicativo (cambiare i riferimenti al service name, al cluster, ai repository, i nomi delle key , region etc), ed è modificato rispetto a quelli presenti online per usare il buildpack di mvn per generare direttamente immagine docker, senza passare per la docker build di un Dockerfile  
  
A questo punto se faccio una push sul repository GitHub, e vado in actions, si è creata la pipeline/action. La posso lanciare manualmente, o tutti i successivi push sul branch indicato (develop) faranno partire il deploy della action su github.  
**NB: l’istanza EC2 deve essere running altrimenti github non la riesce a startare da solo, ma bisogna stoppare i service dei deploy fatti precedentemente a mano se usano stessa porta /risorse complessive del cluster (andare sul servizio e cancellare il servizio, ma non il cluster) prima di fare il deploy con l’action di github (la prima volta, le successive dovrebbero usare lo stesso service)**

**A screenshot of a computer

Description automatically generated**