

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

CORSO DI LAUREA IN INFORMATICA

Tesi di Laurea

Studio del servizio AWS Batch e implementazione in un workflow aziendale

Relatore:

Prof. Giacomo Cabri

Laureando:

Stefano Vezzalini

Anno Accademico 2020-2021

Indice

| | | |
|----------|--|----------|
| 1 | Introduzione | 5 |
| 1.1 | Amazon Web Services | 6 |
| 1.2 | Presentazione del progetto | 6 |
| 1.3 | Obiettivo e risultati attesi | 7 |
| 2 | I servizi utilizzati | 8 |
| 2.1 | AWS Batch | 8 |
| 2.1.1 | I componenti fondamentali | 9 |
| 2.1.1.1 | Jobs | 9 |
| 2.1.1.2 | Job Definition | 9 |
| 2.1.1.3 | Job queues | 10 |
| 2.1.1.4 | Compute Environment | 10 |
| 2.2 | Amazon S3 | 11 |
| 2.2.1 | Classi di storage | 11 |
| 2.3 | AWS Lambda | 11 |
| 2.4 | Amazon EventBridge | 12 |
| 2.5 | Amazon ECR | 12 |
| 2.6 | Amazon SNS | 13 |

| | |
|---|-----------|
| <i>INDICE</i> | 2 |
| 2.6.1 Docker | 14 |
| 2.7 Amazon EFS | 14 |
| 3 L'ambiente di lavoro | 15 |
| 3.1 Creazione dell'ambiente di lavoro Batch | 15 |
| 3.1.1 Creazione di un Compute Environment | 15 |
| 3.1.2 Creazione di una Job Queue | 17 |
| 3.2 I Job | 17 |
| 3.2.1 Il linguaggio di programmazione | 18 |
| 3.2.2 Spring Boot | 18 |
| 3.2.2.1 Maven | 19 |
| 4 Problemi iniziali | 21 |
| 4.1 Problemi affrontati | 21 |
| 4.2 Soluzioni trovate | 22 |
| 4.2.1 Soluzione al problema 1 | 22 |
| 4.2.2 Soluzione ai problemi 2 e 3 | 23 |
| 4.2.3 Soluzione al problema 4 | 23 |
| 4.3 Monitorare il funzionamento dei job | 24 |
| 5 La realizzazione del progetto | 26 |
| 5.1 Unpack Job | 27 |
| 5.1.1 Parti interessanti del codice | 28 |
| 5.1.1.1 Il download dell'oggetto | 28 |
| 5.1.1.2 Submit del job successivo | 30 |
| 5.2 Transform Job | 31 |

| | |
|---|-----------|
| <i>INDICE</i> | 3 |
| 5.3 Compose Job | 32 |
| 5.4 Delivery Job | 32 |
| 5.5 Lambda Function e notifica tramite SNS | 32 |
| 5.5.1 Pubblicazione su SNS | 34 |
| 6 Miglioramento della code quality | 36 |
| 6.1 Creazione di un progetto unico | 36 |
| 6.2 SonarQube e Code Quality | 39 |
| 6.3 Unit Testing e refactoring | 40 |
| 6.3.1 Unit testing con Mockito | 41 |
| 6.4 Rimozione di code smell e vulnerabilità | 43 |
| 6.5 Situazione finale | 44 |
| 7 Continuous Integration | 46 |
| 7.1 Strumenti utilizzati | 46 |
| 7.1.1 Bitbucket | 46 |
| 7.1.2 Bamboo | 47 |
| 7.2 Implementazione | 47 |
| 7.2.1 Modifica del pom e plugin Spotify | 47 |
| 7.2.2 Uso di Bamboo | 49 |
| 7.2.2.1 Initialize | 50 |
| 7.2.2.2 Build | 51 |
| 7.2.2.3 Publish | 51 |
| 8 Considerazioni finali | 52 |
| 8.1 Studio dei costi | 52 |

| | |
|---------------|---|
| <i>INDICE</i> | 4 |
|---------------|---|

| | |
|---|----|
| 8.2 Limitazioni nell'utilizzo di servizi di cloud computing | 54 |
| 8.2.1 Quali leggi applicare? | 55 |
| 8.2.2 Privacy e dati sensibili | 55 |

Capitolo 1

Introduzione

Il progetto che sono andato a svolgere e di cui parlerò più approfonditamente in seguito è stato svolto presso l'azienda Doxee Spa. Doxee è un'azienda specializzata nell'offerta di prodotti in ambito Customer Communications Management.(CCM), Digital Customer Experience e Dematerializzazione; aiuta i suoi clienti ad innovare e modernizzare la relazione con i propri clienti andando a trasformare la Customer Experience ponendosi tra un'azienda fornitrice di un servizio e il cliente finale.

Doxee offre diverse linee di prodotto, quella che ho avuto modo di vedere più da vicino andandomi ad inserire nel team di ricerca e sviluppo è chiamata **dx** o **document experience**, ovvero la linea dedicata alla produzione e alla distribuzione documentale a partire da dati grezzi.

1.1 Amazon Web Services

L'azienda, come molte altre in tutto il mondo, sfrutta gli Amazon Web Services per alcuni dei suoi processi produttivi.

Amazon Web Services è un'azienda di proprietà del gruppo Amazon che fornisce servizi di cloud computing su un'omonima piattaforma. I servizi offerti, che spaziano dall'essere puramente assistenti di calcolo a basi di dati fino ad arrivare al machine learning, consentono di mantenere alti livelli di sicurezza, high availability e ridondanza. Il costo finale di questi servizi deriva in tutti i casi dalla combinazione del tipo, della quantità e dal tempo di utilizzo delle risorse utilizzate.

Ormai sempre più aziende stanno sfruttando questo tipo di servizi piuttosto che implementarli internamente: il risparmio è quasi sempre considerevole, viene rimossa quasi totalmente la necessità di gestire i server e il rischio di guasti che possano bloccare la produzione diventa quantomeno remoto.

1.2 Presentazione del progetto

Il progetto che sono andato a realizzare ha consistito nella creazione di una POC (proof of concept) che permettesse di testare alcuni processi produttivi utilizzando il servizio AWS Batch¹.

¹<https://aws.amazon.com/it/batch/>

L'idea alla base era quella di prendere un workflow tipico e spostarne le varie parti su AWS Batch integrandolo anche con altri servizi Amazon.

La situazione tipo su cui si è ragionato era quella in cui uno file zip contenente un file di testo di lunghezza variabile e in un formato grezzo venisse caricato su un bucket S3, questo upload sarà poi l'evento scatenante che andrà a far partire il primo job (questo è il nome che la documentazione AWS usa per rappresentare un'unità di lavoro all'interno di una coda su AWS Batch) del workflow visualizzabile nella figura 1.1; ogni job si occuperà poi di far partire il job successivo inviando i giusti parametri fino ad arrivare all'ultimo che si occuperà della consegna.

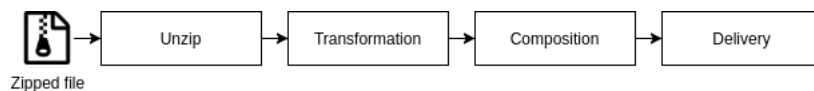


Figura 1.1: Workflow tipico da spostare su AWS Batch

1.3 Obiettivo e risultati attesi

L'obiettivo del progetto è stato quello di verificare se, sfruttando AWS batch al posto dei servizi interni che vengono usati al momento, si potesse ottenere un risultato con performance paragonabili se non migliori e allo stesso tempo ad un costo vantaggioso per l'azienda.

Capitolo 2

I servizi utilizzati

Di seguito, un riepilogo sui servizi utilizzati durante lo svolgimento del progetto.

2.1 AWS Batch

AWS Batch¹ è un servizio Amazon rilasciato nel 2016 che permette agli sviluppatori di eseguire in maniera semplice centinaia di lavori di elaborazione in batch su AWS. In informatica per batch processing si intende l'esecuzione di tasks senza che sia necessaria l'interazione dell'utente o che possono essere programmati per essere eseguiti quando le risorse lo permettono. AWS Batch si occupa del provisioning dinamico della quantità e del tipo di risorse ottimali per il lavoro da eseguire facendo pagare all'utente solo per le risorse che vengono create per archiviare ed eseguire i lavori in batch.

¹<https://aws.amazon.com/it/batch/>

2.1.1 I componenti fondamentali

2.1.1.1 Jobs

I job sono l'unità di lavoro di AWS Batch e possono essere invocati come applicazioni "containerizzate" in esecuzione su istanze EC2 o tramite Fargate. Ogni job fa riferimento ad una container image (e.g.: una Docker Image) che definisce come l'applicazione containerizzata verrà eseguita. È possibile sottomettere una grande quantità di job semplici ed indipendenti tra di loro.

2.1.1.2 Job Definition

Ogni job deve essere in relazione ad una job definition che specifica in che modo questo verrà eseguito. Una job definition viene generata a partire da diversi parametri che possono anche essere sovrascritti a runtime.

Gli attributi specificati in una job definition includono:

- Quale Docker image utilizzare con il container nel job che vogliamo andare a sottomettere;
- Quante virtual CPUs e quanta memoria usare all'interno del container;
- Il comando che il container deve eseguire quando viene avviato;
- Se presenti, quali variabili d'ambiente passare al container quando viene avviato;
- Quale IAM Role il job deve usare, ovvero quali permessi AWS il nostro job possiederà.

2.1.1.3 Job queues

Ogni job che si vuole eseguire viene inserito all'interno di una job queue dove rimarrà fino a quando non potrà essere eseguito all'interno di un compute environment.

Un account AWS può creare diverse job queue. Ad esempio si può utilizzare una queue che sfrutta un'istanza (EC2 o Fargate) On-demand per job ad alta priorità ed un'altra che usa istanze Spot per job di più bassa priorità.

Ad ogni job queue viene assegnato un valore che rappresenta la **priorità** che serve allo scheduler per decidere quale job in quale queue deve essere eseguito prima.

2.1.1.4 Compute Environment

Ogni Job queue è mappata su una o più compute environment. I compute environment contengono l'istanza del container che verrà utilizzato per eseguire i batch job containerizzati.

Un compute environment può essere **managed** oppure **unmanaged**: nei primi AWS Batch aiuta l'utente a gestire la capacità e i tipi di istanza delle risorse di calcolo all'interno dell'ambiente. In figura 2.1 si può vedere la struttura semplificata di AWS Batch e come si pongono i vari componenti.

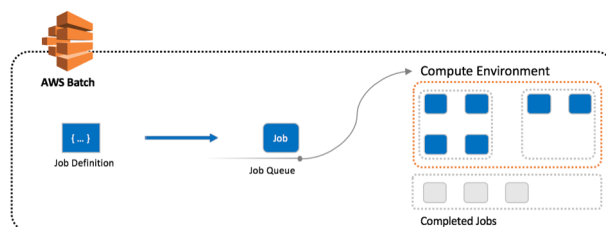


Figura 2.1: Struttura semplificata di AWS Batch

2.2 Amazon S3

Amazon S3² è un servizio di storage di oggetti che offre scalabilità, disponibilità dei dati, sicurezza ed alte prestazioni. È applicabile ad una vasta gamma di casi d'uso come siti web, applicazioni web, backup e ripristino.

S3 garantisce una durabilità dei dati che rasenta il 100% grazie ad appositi meccanismi di mirroring dei dati su più sistemi.

2.2.1 Classi di storage

Amazon S3 permette di utilizzare diverse classi di storage in base all'utilizzo che l'utente o l'azienda deve farne e in modo da ottimizzare al massimo la spesa.

2.3 AWS Lambda

AWS Lambda³ è un servizio di elaborazione *serverless* che si occupa di eseguire il codice scritto da un utente occupandosi inoltre di gestire automaticamente le risorse di calcolo in uso al suo posto e facendo pagare solo in base all'utilizzo effettivo che si fa del servizio.

AWS Lambda permette di estendere gli altri servizi AWS con logica personalizzata oppure di creare interamente nuovi servizi di back-end che sfruttino i vantaggi offerti da AWS.

²<https://aws.amazon.com/it/s3/>

³<https://aws.amazon.com/it/lambda/>

Lambda esegue il codice in maniera automatica a seguito di eventi multipli, come richieste HTTP effettuate tramite API, modifiche agli oggetti all'interno di un bucket S3, aggiornamenti su un database e altro ancora.

Quando si decide di creare una funzione lambda è possibile selezionare diverse opzioni:

- Crearla da zero selezionando un runtime tra i diversi disponibili (e.g.: Python, Node.js, Go...) e scrivendola sull'editor integrato;
- Crearla partendo da un'insieme di codici d'esempio detti Piani;
- Selezionando un'immagine dal container da distribuire per la funzione;
- Distribuendo un'applicazione Lambda d'esempio da AWS Serverless Application Repository.

2.4 Amazon EventBridge

Amazon EventBridge⁴ è un servizio bus di eventi serverless che semplifica e permette la creazione di applicazioni basate su eventi permettendo di far comunicare più servizi AWS tra di loro.

2.5 Amazon ECR

Amazon Elastic Container Register⁵ è un registro del container totalmente managed che semplifica e permette l'archiviazione, la gestione, la condivisio-

⁴<https://aws.amazon.com/it/eventbridge/>

⁵<https://aws.amazon.com/it/ecr/>

ne e l'implementazione di immagini di container (e.g.: immagini Docker) e artefatti ovunque.

2.6 Amazon SNS

Amazon SNS⁶ è un servizio di messaggistica per la comunicazione application-to-person e application-to-application.

I messaggi prodotti dal nostro sistema vengono inviati ad Amazon SNS che poi si occupa di distribuirli ai cosiddetti Subscribers e che li riceveranno attraverso una scelta di canali come ad esempio le email. In figura 2.2 un'illustrazione che spiega il funzionamento del servizio.

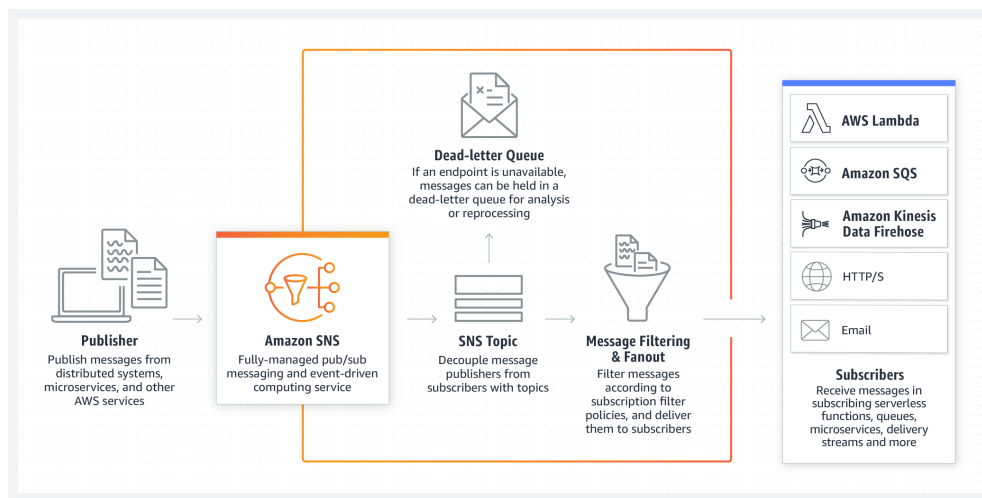


Figura 2.2: Funzionamento di Amazon SNS

⁶<https://aws.amazon.com/it/sns/>

2.6.1 Docker

Docker⁷ è un progetto open-source che automatizza il deployment di applicazioni all'interno di container andando ad aggiungere un'ulteriore livello di astrazione grazie alla virtualizzazione a livello di sistema operativo di Linux.

Docker viene utilizzato per pacchettizzare un'applicazione insieme alle sue dipendenze in un container virtuale molto leggero che possa essere eseguito su ogni sistema operativo, sia esso Linux, Windows o macOS. Questo permette di realizzare applicazioni che possano eseguire in una vasta gamma di situazioni, come nel caso in cui si stiano utilizzando servizi di cloud computing pubblici o privati.

Docker ha portato enormi vantaggi nel mondo dell'informatica a partire dal suo rilascio nel 2013; la possibilità di pacchettizzare tutto in un unico container ha semplificato di molto l'implementazione di applicazioni anche su sistemi molto eterogenei tra loro.

2.7 Amazon EFS

Amazon EFS⁸(Elastic File System) è un servizio che permette di creare e configurare file system condivisi per i servizi di calcolo AWS. La gestione del file system è lasciata quasi totalmente ad Amazon che ne riduce o aumenta la dimensione in base a quanti file sono memorizzati e si occupa di implementazione, patching e manutenzione.

⁷<https://www.docker.com/>

⁸<https://aws.amazon.com/it/efs/>

Capitolo 3

L'ambiente di lavoro

In questo capitolo verrà spiegato come è stato preparato l'ambiente di lavoro su cui svolgere il progetto andando a concentrarsi sul settaggio delle varie componenti AWS Batch e sulla versione di Java utilizzata.

3.1 Creazione dell'ambiente di lavoro Batch

3.1.1 Creazione di un Compute Environment

Per prima cosa è stato importante realizzare un ambiente batch su cui andare a sviluppare il nostro progetto. Nel mio caso ho utilizzato direttamente l'interfaccia web di AWS per sfruttare tutti i servizi necessari ma è possibile fare tutto anche tramite la shell AWS.

Parametri fondamentali da settare sono:

- Il tipo di compute environment, ovvero se vogliamo che sia managed oppure unmanaged;
- Il modello di provisioning, con la possibilità di scegliere tra:
 - **Fargate**: permette di eseguire processi AWS Batch in maniera *serverless* senza il bisogno di gestire istanze EC2;
 - **Fargate spot**: permette di eseguire i task sulle risorse libere in un dato momento nel cloud AWS permettendo di risparmiare ulteriormente; in un qualsiasi momento il task può però venire interrotto (anticipato due minuti prima da una notifica);
 - **On Demand**: Istanza di EC2 che viene fatturata al secondo, è di proprietà dello sviluppatore che però dovrà anche gestirne tutti gli aspetti e mantenerla;
 - **Spot**: Stesso principio di Fargate spot ma su istanze EC2.
- Numero massimo di vCPUs (CPU virtuali);
- Numero minimo di vCPUs (nel caso si decidesse di non usare un modello di provisioning di tipo Fargate);
- Sottoreti VPC su cui avviare le risorse di calcolo.

Nel mio caso si è scelto di realizzare un compute environment managed di tipo Fargate a cui è stato assegnato un numero massimo di 4 vCPU, come è possibile vedere in figura 3.1 , presa direttamente dall'interfaccia web di configurazione.

Create compute environment

Compute environment configuration

Compute environment type

☒ **Managed**
AWS scales and configures your instances for you.

☐ **Unmanaged**
You control and manage the instance configuration, provisioning, and scaling.

Compute environment name

Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

☒ **Enable compute environment**

Service role

Batch service-linked role

AWS Batch uses a service-linked role that includes all the permissions Batch requires to call other AWS services on your behalf. If you do not have a Batch service-linked role, we will create it for you.

Instance configuration

Save up to 90% on fully managed batch processing
Learn how the Spot Capacity Optimized allocation strategy can dramatically lower the cost of your AWS Batch compute environment. [Learn more](#)

Provisioning model

☒ **Fargate**
Allows AWS Batch to run containers without having to manage servers or clusters of Amazon EC2 instances.

☐ **Fargate Spot**
Fargate Spot allows you to run interruption tolerant AWS Batch jobs at up to a 70% discount off the Fargate price.

☐ **On-demand**
EC2 instances that are billed per second.

☐ **Spot**
Save money by using Spot instances but be aware your instances can be interrupted with a two minute notification when EC2 needs the capacity back.

Maximum vCPUs

256

Maximum vCPUs you can have.

Figura 3.1: Interfaccia web per la creazione di un compute environment

3.1.2 Creazione di una Job Queue

La creazione di una job queue è molto facile, richiede solo di assegnarle un nome ed una priorità che ho lasciato al valore di default (1) dal momento che avrei lavorato solo con questa.

3.2 I Job

Il workflow su cui si è voluto realizzare il progetto era formato da 4 job, in questa sezione andrò a spiegare il linguaggio di programmazione utilizzato

ed altre specifiche importanti.

3.2.1 Il linguaggio di programmazione

Il linguaggio di programmazione utilizzato per programmare la logica di ogni job è Java, utilizzando Amazon Corretto 11 come ambiente di sviluppo, ovvero una distribuzione di OpenJDK supportata a lungo termine fornita da Amazon.

Amazon supporta costantemente questa distribuzione fornendo aggiornamenti trimestrali per garantire sempre le migliori performance e soluzioni per problemi relativi alla sicurezza di importanza critica nello sviluppo di applicazioni aziendali.

Utilizzando Corretto si è sicuri al 100% che non ci saranno problemi a lavorare con i diversi servizi Amazon.

3.2.2 Spring Boot

Per lo sviluppo si è usato il Java Spring Framework (**Spring Boot**)¹, in quanto l'azienda ne fa uso per lo sviluppo di tutti i suoi **micro servizi**.

Cos'è un micro servizio? Si dice micro servizio un tipo di architettura che permette al programmatore di sviluppare e rilasciare servizi indipendenti tra di loro. Ogni servizio in esecuzione ha il proprio processo, in questo modo si ottiene un modello molto leggero a supporto delle applicazioni aziendali.

¹<https://spring.io/projects/spring-boot>

3.2.2.1 Maven

Spring Boot dà la possibilità di utilizzare Maven² come strumento di gestione del progetto. Maven permette di gestire le build del progetto, il testing, il deployment e le dipendenze. I progetti Maven vengono configurati utilizzando un Project Object Model (POM) memorizzato in un file `pom.xml`. Di seguito un `pom.xml` d'esempio:

```
<project>
  <!-- model version is always 4.0.0 for Maven 2.x POMs -->
  <modelVersion>4.0.0</modelVersion>

  <!--
    project coordinates, i.e. a group of values which uniquely
    identify this project
  -->
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>
  <!-- library dependencies -->
  <dependencies>
    <dependency>
      <!-- coordinates of the required library -->
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <!-- this dependency is only used for running and compiling tests -->
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Per quanto riguarda la gestione delle dipendenze, di default viene utilizzata la Maven Central Repository³ per cercare librerie ma uno sviluppatore

²<https://maven.apache.org/>

³<https://search.maven.org/>

può decidere di impostare anche altre repository (per esempio quelle private dell'azienda in cui lavora). Nel mio caso ho avuto bisogno di sfruttare dipendenze esterne per interfacciarmi in modo più rapido con le API dei diversi servizi Amazon; tutte le librerie facenti parte dell'SDK (Software Development Kit) fornito da Amazon possono essere trovate nella repository Maven di default.

Capitolo 4

Problemi iniziali

Prima di andare a spiegare la realizzazione dei 4 job vediamo alcuni problemi preliminari e le soluzioni trovate.

4.1 Problemi affrontati

Essendo i 4 job batch separati e indipendenti tra di loro era necessario trovare un modo per metterli in comunicazione, in particolare fare in modo che:

1. L'upload di un oggetto su un bucket S3 faccia partire la catena di job con il file appena caricato come input;
2. Un job n riceva i parametri corretti dal job $n - 1$;
3. Un job n , quando terminato, triggeri la partenza del job successivo;
4. I file di output di un certo job devono essere memorizzati da qualche parte in modo che il job successivo possa recuperarli ed utilizzarli come input.

4.2 Soluzioni trovate

4.2.1 Soluzione al problema 1

Per ottenere il risultato desiderato è stato necessario appoggiarsi ad un servizio della suite AWS utile proprio in situazioni come questa: EventBridge. EventBridge permette di definire un evento sorgente tramite un set di regole detto Event Pattern settabile sia tramite interfaccia grafica sia andando a scriverlo in formato JSON. In figura 4.1, l'event pattern che permette di catturare l'upload di un file all'interno di un bucket S3.

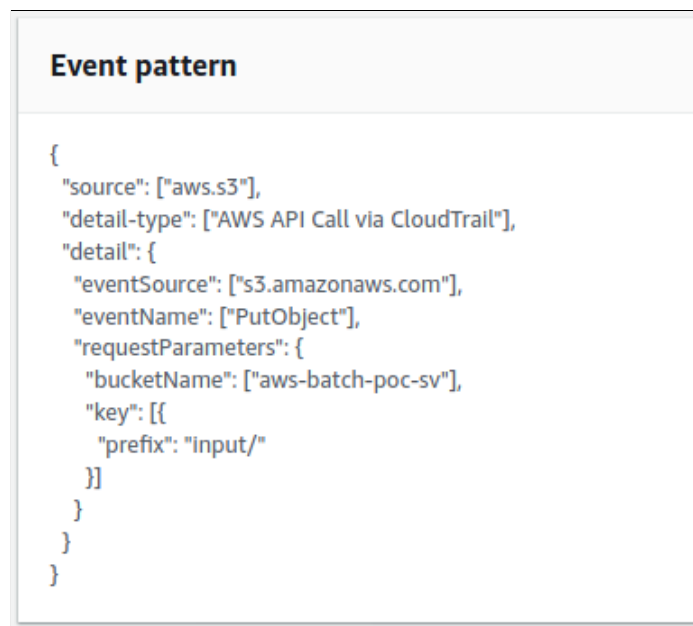


Figura 4.1: Event pattern in formato JSON per catturare un evento su bucket S3

Ne spiego brevemente le parti fondamentali:

- **source**: indica il servizio AWS da monitorare;

- **eventName**: indica quale evento monitorare sul bucket, in questo caso l'upload che su S3 viene chiamato **PutObject**;
- **bucketName**: indica quale bucket monitorare;
- **prefix**: segnala all'event pattern di considerare solo gli oggetti con suffisso "input/", ovvero tutti quelli che sono stati caricati nella sottocartella input del bucket.

EventBridge permette poi di usare AWS Batch come Target dell'event pattern andando anche ad indicare quali parametri inviare al job, la modalità in cui questi parametri verranno sfruttati si vedrà meglio più avanti.

4.2.2 Soluzione ai problemi 2 e 3

Essendo intrinsecamente necessario che ogni catena di job lavori su un solo file, per risolvere questo problema ho fatto uso del SDK Java fornito da Amazon. Al termine di un job è il job stesso che genera dinamicamente la struttura del job successivo e lo immette nella queue andando anche ad indicare come parametro su quali file o cartelle lavorare. Anche questo aspetto verrà approfondito meglio più avanti.

4.2.3 Soluzione al problema 4

Per risolvere questo problema è stato usato Amazon EFS, un servizio che permette di creare file system in maniera serverless e di usarli in appoggio ad altri servizi. Nel mio caso il file system creato è stato montato su ognuno dei 4 job che in questo modo potevano avere uno spazio condiviso in cui trovare

i file e le cartelle su cui lavorare, i path per questi ultimi vengono ricevuti come parametri nel modo spiegato nella scorsa sezione. In figura 4.2 si può osservare la panoramica generale del file system creato.

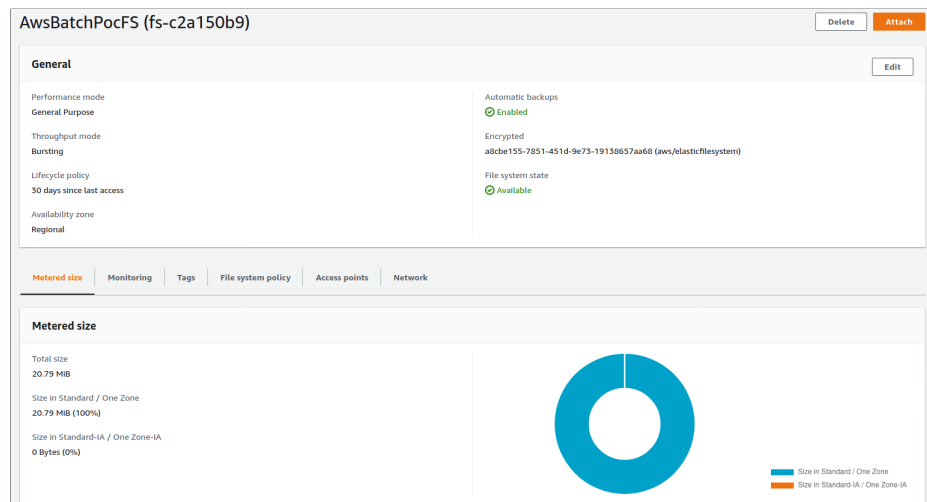


Figura 4.2: Pannello di controllo per il file system per monitorare lo spazio occupato, gestire i permessi ed altro

4.3 Monitorare il funzionamento dei job

Trovate le diverse soluzioni per permettere ai job di collaborare efficacemente pur rimanendo indipendenti è risultato necessario avere un modo per poter controllare in modo veloce che l'output di ogni job fosse quello atteso. Per fare questo si è deciso di avviare una macchina EC2 in cui montare il file system usato dai job per monitorarlo tramite comandi bash.

Cos'è una macchina EC2? Amazon EC2¹ è un servizio Amazon che permette di creare macchine virtuali di diverso tipo e con la possibilità di

¹<https://aws.amazon.com/it/ec2>

scegliere tra un'ampia gamma di sistemi operativi.

Essendo necessaria solo per il monitoraggio si è scelto di usare un'istanza di tipo t2.micro che grazie alla sua bassa potenza di calcolo permette di mantenere i costi contenuti, costi che erano ridotti ulteriormente dal fatto che la macchina veniva avviata solo quando necessario. In figura 4.3 vengono mostrate le caratteristiche e i casi d'uso delle diverse istanze EC2.

| Istanza | vCPU* | Crediti CPU/ora | Mem (GiB) | Archiviazione | Prestazioni di rete |
|------------|-------|-----------------|-----------|---------------|---------------------|
| t2.nano | 1 | 3 | 0,5 | Solo EBS | Basso |
| t2.micro | 1 | 6 | 1 | Solo EBS | Da basse a moderate |
| t2.small | 1 | 12 | 2 | Solo EBS | Da basse a moderate |
| t2.medium | 2 | 24 | 4 | Solo EBS | Da basse a moderate |
| t2.large | 2 | 36 | 8 | Solo EBS | Da basse a moderate |
| t2.xlarge | 4 | 54 | 16 | Solo EBS | Moderate |
| t2.2xlarge | 8 | 81 | 32 | Solo EBS | Moderate |

Tutte le Istanze possiedono le seguenti specifiche:

- [Intel AVX†](#), [Intel Turbo†](#)
- t2.nano, t2.micro, t2.small, t2.medium hanno fino a 3,3 GHz di processore Intel scalabile
- t2.large, t2.xlarge e t2.2xlarge hanno fino a 3,0 GHz di processore Intel scalabile

Casi d'uso

Siti Web e applicazioni Web, ambienti di sviluppo, server di compilazione, repository di codice, microservizi, ambienti di test e di gestione temporanea e applicazioni line-of-business.

Figura 4.3: Caratteristiche delle diverse istanze della famiglia T2

Capitolo 5

La realizzazione del progetto

In questo capitolo si parlerà più nel dettaglio di come sono stati realizzati i quattro job; la struttura completa e ad alto livello del progetto è visibile in figura 5.1.

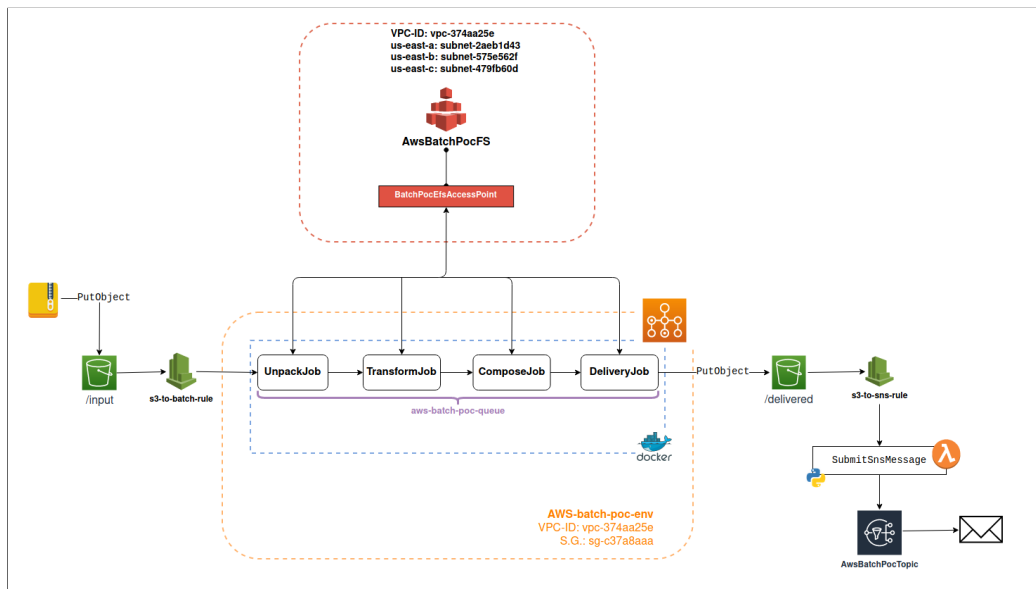


Figura 5.1: Design completo e ad alto livello del progetto

Vediamo ora in che modo è stato realizzato.

5.1 Unpack Job

Il ruolo dell’Unpack Job è molto semplice: scaricare un file zip da un bucket S3, decomprimerlo e salvarlo sul file system EFS. Vediamo in figura 5.2 la job definition, le altre tre seguono praticamente la stessa struttura.

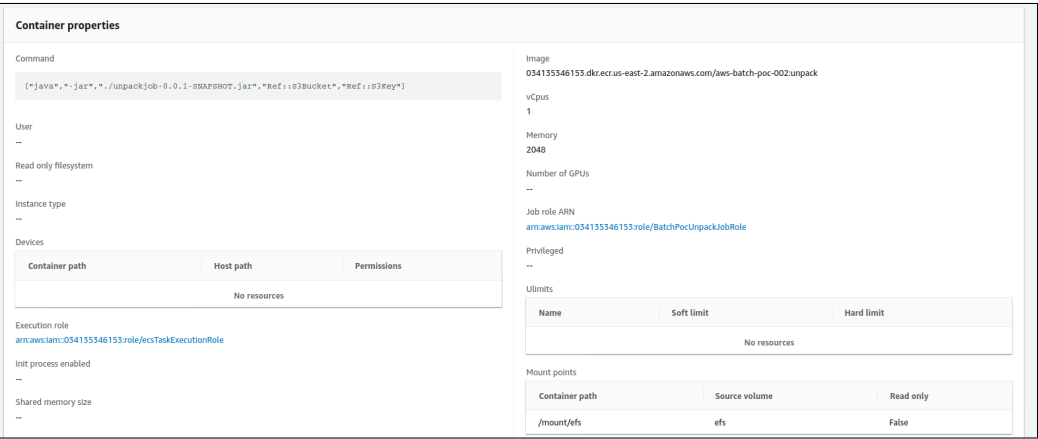


Figura 5.2: Job definition per l’Unpack Job



Figura 5.3: Dockerfile per l’Unpack Job

Approfondiamone meglio alcuni dettagli:

- La sezione **Command** in alto a sinistra rappresenta il comando CMD scritto nel dockerfile che possiamo osservare in figura 5.3 e indica il comando bash da lanciare nel momento in cui viene creato il container virtuale a partire dall’immagine Docker;

- Il prefisso `Ref::` serve ad indicare che quel valore va sostituito con un parametro ricevuto da un altro servizio, in questo caso da `EventBridge` ma può anche essere ricevuto da un altro job.
- L'**Execution Role** in basso a sinistra delinea quali permessi ha il job nei confronti di AWS, in questo caso quello di interfacciarsi con S3 e di lanciare un nuovo job;
- La sezione **Image** riporta il link all'immagine Docker corretta su ECR;
- La sezione **Mount points** riporta quanti e quali file system verranno montati sul container, in questo caso solamente quello generato tramite EFS di cui si è parlato nel capitolo precedente e che permette ai job di interoperare;
- Il comando `WORKDIR` del Dockerfile imposta sostanzialmente l'entry point in cui l'immagine si troverà una volta avviata;
- Il comando `COPY` prende il jar compilato dal comando `mvn clean package` e lo "importa" nel container virtuale creato dall'immagine mettendolo nella working directory.

5.1.1 Parti interessanti del codice

Andiamo a vedere le parti del codice maggiormente degne di nota.

5.1.1.1 Il download dell'oggetto

In figura 5.4 si può osservare la parte di codice dedicata al download dell'oggetto, da notare l'utilizzo di un algoritmo di Exponential Backoff per gestire

```
30 public InputStream downloadObject(String s3BucketName, String s3ObjectKey) {
31
32     S3Object fullObject = null;
33
34     int retries = 0;
35     boolean retry = false;
36     final int MAX_RETRIES = 3;
37     final int MAX_WAIT_INTERVAL = 30000;
38
39     do {
40         try {
41             long waitTime = Math.min(TimeUtilities.getWaitTimeExp(retries), MAX_WAIT_INTERVAL);
42             log.info("[LOG] waitTime: " + waitTime);
43             Thread.sleep(waitTime);
44
45             fullObject = s3Client.getObject(new GetObjectRequest(s3BucketName, s3ObjectKey));
46             log.info("Downloaded " + fullObject.getKey());
47
48             return fullObject.getObjectContent();
49         } catch (AmazonServiceException e) {
50             log.error("AmazonServiceException while downloading object from S3", e);
51             retry = true;
52         } catch (InterruptedException e) {
53             log.error("InterruptedException while downloading object from S3", e);
54             Thread.currentThread().interrupt();
55         }
56     } while (retry && (retries++ < MAX_RETRIES));
57     return null;
58 }
59 }
```

Figura 5.4: Metodo per il download dell'oggetto da S3

altri tentativi nel caso in cui il primo fallisca per esempio per un problema di rete.

Algoritmo di Exponential Backoff: si definisce un numero massimo di tentativi e ogni volta che si presenta un errore l'attesa tra un tentativo e l'altro cresce in modo esponenziale fino a quando non l'errore non si ripresenta o si incontra il valore dato da `MAX_RETRIES`. La documentazione AWS suggerisce di gestire in questo modo questo genere di errori che possono essere dati da problemi di comunicazione di rete momentanei.

5.1.1.2 Submit del job successivo

Ogni job (tranne l'ultimo) è incaricato anche di far partire il job successivo alla fine, per farlo ha bisogno di definire tre aspetti:

- Le proprietà del container da avviare, quindi il comando (che nella figura 5.2 è riportato nella sezione Command) e le risorse da utilizzare che per ogni container del progetto sono sempre state 1 vCPU e 2GB di memoria, questo aspetto è visibile in figura 5.5;
- I parametri da inviare al job successivo, visibili in figura 5.6;
- Le proprietà della job request da inviare, visibili in figura 5.7.

```
25 private ContainerOverrides nextJobContainerBuilder() {  
26  
27     List<String> nextJobCommands =  
28         Arrays.asList(  
29             "java",  
30             "-jar",  
31             "composejob-0.0.1-SNAPSHOT.jar",  
32             "Ref::S3Bucket",  
33             "Ref::dtOutputPath");  
34  
35     List<ResourceRequirement> nextJobResources =  
36         Arrays.asList(  
37             (new ResourceRequirement()).withType("VCPU").withValue("1"),  
38             (new ResourceRequirement()).withType("MEMORY").withValue("2048"));  
39  
40     return new ContainerOverrides()  
41         .withCommand(nextJobCommands)  
42         .withResourceRequirements(nextJobResources);  
43 }
```

Figura 5.5: Metodo definire le impostazioni per il prossimo job del workflow


```
45 @ private Map<String, String> nextJobParamsBuilder(String dtOutputFilePath, String s3BucketName) {  
46  
47     Map<String, String> nextJobParams = new HashMap<>();  
48     nextJobParams.put( k: "dtOutputPath", dtOutputFilePath);  
49     nextJobParams.put( k: "S3Bucket", s3BucketName);  
50  
51     return nextJobParams;  
52 }
```

Figura 5.6: Metodo per definire i parametri da inviare al job successivo

```
54 public SubmitJobRequest submitJobRequestBuilder(String dtOutput, String s3BucketName) {  
55  
56     ContainerOverrides nextJobContainerOverrides = nextJobContainerBuilder();  
57     Map<String, String> nextJobParams = nextJobParamsBuilder(dtOutput, s3BucketName);  
58  
59     return (new SubmitJobRequest())  
60         .withJobDefinition(  
61             "arn:aws:batch:us-east-2:034135346153:job-definition/aws-batch-compose-job")  
62         .withJobQueue("arn:aws:batch:us-east-2:034135346153:job-queue/aws-batch-poc-queue")  
63         .withJobName("compose-job")  
64         .withParameters(nextJobParams)  
65         .withContainerOverrides(nextJobContainerOverrides);  
66 }
```

Figura 5.7: Metodo per definire le proprietà del job successivo come la job definition, la job queue in cui inserirlo, etc.

Queste operazioni avvengono in maniera simile anche per Transform e Compose Job.

5.2 Transform Job

Il Transform Job si occupa di prendere i dati grezzi del file precedentemente unzippato e trasformarli in un formato apposito che sarà utile nella fase successiva.

5.3 Compose Job

Il Compose Job prende i dati trasformati ed effettua un'operazione di compositing che ha come output un pdf formattato secondo il template dell'azienda. Questo file ora è pronto ad essere caricato nuovamente su S3 per essere spedito al cliente finale.

5.4 Delivery Job

Così come l'Unpack Job anche questo ha un compito molto semplice: prendere il pdf precedentemente generato e caricarlo su S3 da cui partirà la vera e propria fase di delivery.

5.5 Lambda Function e notifica tramite SNS

Al fine di testare progetto ho creato un nuovo topic SNS a cui mi sono iscritto visibile in figura 5.8.

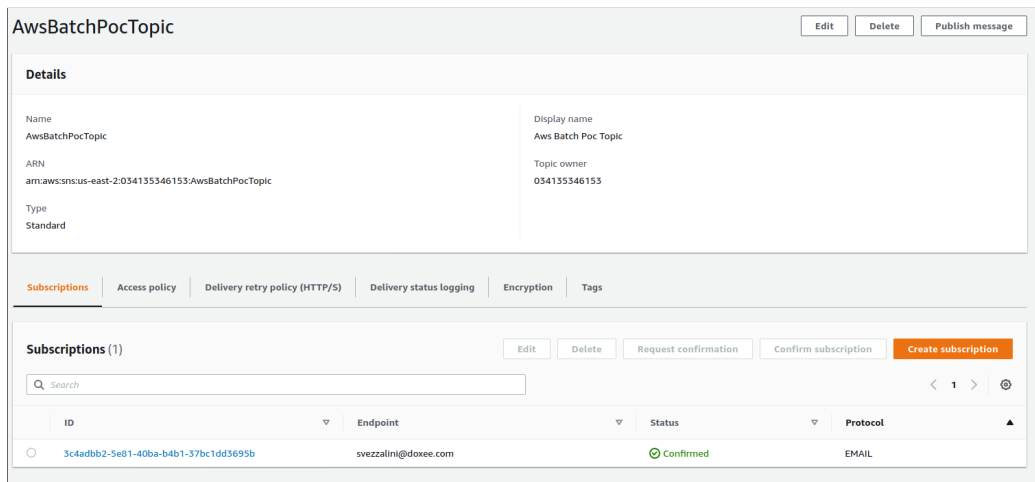


Figura 5.8: Topic SNS realizzato per il progetto

Un topic Amazon SNS è un punto di accesso logico che funge da canale di comunicazione e permette di raggruppare diversi endpoint come SMS, AWS Lambda, HTTPS e Email. Per funzionare il topic ha bisogno anche di subscription, ovvero utenti (rappresentati da mail, numeri di cellulare, etc.) che possano ricevere le notifiche attraverso l'apposito canale di iscrizione. Nel mio caso ho creato una subscription con la mia email al topic creato (visibile in basso in figura 5.8), il processo di creazione di una subscription tramite interfaccia web è visibile in figura 5.9.

Create subscription

Details

Topic ARN
Q arn:aws:sns:us-east-2:034135346153:AwsBatchPocTopic X

Protocol
The type of endpoint to subscribe

Select protocol ▲

- Amazon Kinesis Data Firehose
- Amazon SQS
- AWS Lambda
- Email
- Email-JSON
- HTTP
- HTTPS
- SMS

Figura 5.9: Interfaccia web per la creazione di una subscription

5.5.1 Pubblicazione su SNS

Sfruttando di nuovo EventBridge in modo simile a come è stato utilizzato nella prima fase del progetto, ho creato una nuova regola che, ogni volta in cui un oggetto viene caricato nel bucket S3 nella sotto-cartella "delivered", triggeri una Lambda Function che si occupi di pubblicare sul topic un messaggio che permetta agli utenti iscritti di ricevere una mail con un link per visualizzare il file. In figura 5.10 si può osservare il codice della lambda function che si occupa di quest'ultimo aspetto.

```
1 import json
2 import boto3
3
4 def lambda_handler(event, context):
5     client = boto3.client('sns')
6     params = event
7
8     topic_arn = 'arn:aws:sns:us-east-2:034135346153:AwsBatchPocTopic'
9     subject = f'A new file has been uploaded to {params["S3Bucket"]}'
10
11     file_url = f'https://{params["S3Bucket"]}.s3.us-east-2.amazonaws.com/{params["S3Key"]}'
12     message = f'check the new file at {file_url}'
13
14     response = client.publish(
15         TopicArn=topic_arn,
16         Subject=subject,
17         Message=message
18     )
19
20     return {
21         'response' : response,
22         'statusCode' : 200,
23     }
```

Figura 5.10: Lambda Function Python per la pubblicazione del messaggio su S3

Per interfacciarsi con AWS tramite python viene utilizzata la libreria Boto3¹ che permette di usare l'SDK AWS per interfacciarsi con i diversi servizi come SNS.

¹<https://boto3.amazonaws.com/v1/documentation/api/latest/index.html>

Capitolo 6

Miglioramento della code quality

Dopo aver terminato quella che può essere chiamata “prima stesura” dei 4 job sono stato introdotto al concetto di code quality.

6.1 Creazione di un progetto unico

Prima di parlare di come è stata migliorata la code quality e della creazione degli unit test è bene fare una piccola digressione su come si è evoluta la struttura del progetto. Come già spiegato in precedenza i 4 job sono tutti progetti Spring Boot che sfruttano Maven come strumento di gestione, successivamente, però, si è deciso di spostare questi 4 progetti separati in un unico grande progetto andando a sfruttare il sistema gerarchico di Maven e dei suoi file `pom.xml`.

Il cambio di struttura porta diversi vantaggi: in primis è possibile eseguire in un solo comando tutte le operazioni legate al testing e alla continuous integration (di cui si parlerà nel dettaglio nel capitolo 8) e permette, inoltre, di ridurre le ripetizioni nel caso di dipendenze condivise tra più job. Come possiamo osservare in figura 6.1, ora il progetto segue una struttura gerarchica in cui ogni `pom.xml` ha il suo ruolo ben preciso.

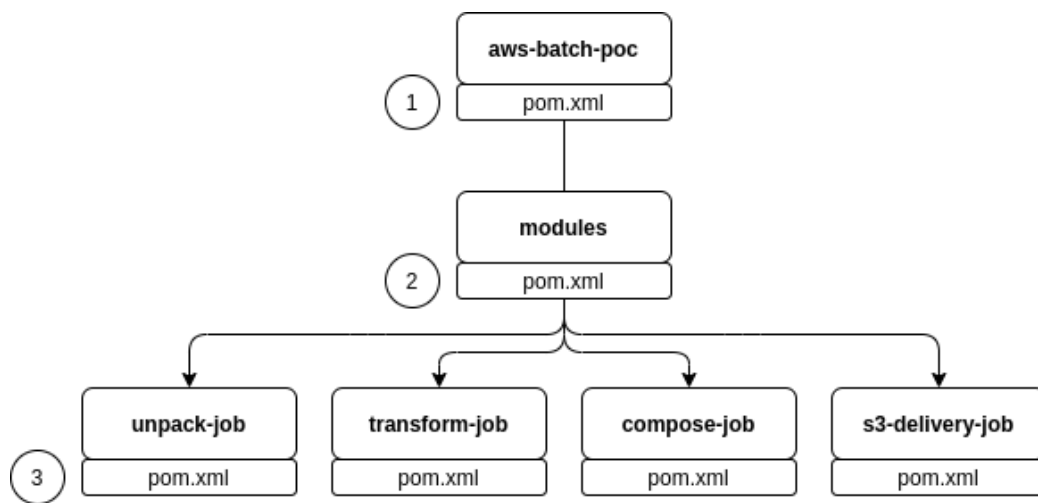


Figura 6.1: Nuova struttura gerarchica del progetto

- Il pom di primo livello definisce tutte le dipendenze condivise tra i quattro job e le informazioni utili alla continuous integration;
- Il pom di secondo livello funge da collegamento tra quello di primo e quello di terzo livello andando ad indicare quali moduli debbano ereditare le dipendenze e le informazioni definite nel livello superiore;
- I 4 pom di terzo livello (uno per ogni job) aggiungono alcune dipendenze necessarie solo al singolo job e aggiungono alcune informazioni necessarie alla continuous integration.

In figura 6.2 possiamo vedere una parte del pom di primo livello in cui vengono indicate alcune dipendenze del progetto.

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <artifactId>spring-boot-starter</artifactId>
      <groupId>org.springframework.boot</groupId>
      <version>${maven-spring.version}</version>
    </dependency>

    <dependency>
      <artifactId>spring-boot-starter-test</artifactId>
      <groupId>org.springframework.boot</groupId>
      <version>${maven-spring.version}</version>
      <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-batch -->
    <dependency>
      <artifactId>aws-java-sdk-batch</artifactId>
      <groupId>com.amazonaws</groupId>
      <version>${aws-batch-sdk.version}</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-s3 -->
    <dependency>
      <artifactId>aws-java-sdk-s3</artifactId>
      <groupId>com.amazonaws</groupId>
      <version>${aws-s3.version}</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
    <dependency>
      <artifactId>commons-io</artifactId>
      <groupId>commons-io</groupId>
      <version>${commons-io.version}</version>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Figura 6.2: Snippet preso dal pom.xml di primo livello in cui vengono indicate alcune delle dipendenze condivise tra più job

6.2 SonarQube e Code Quality

Per **code quality** (detta anche comunemente *software quality*) si intende quanto la struttura del software realizzato va incontro a requisiti non funzionali che sono di supporto alla consegna di requisiti funzionali come ad esempio la robustezza del codice, la manutenibilità e la sicurezza.

Per migliorare la code quality l'azienda fa uso di **SonarQube**¹. SonarQube è una piattaforma open-source per eseguire l'ispezione continua della code quality in modo tale da generare revisioni in modo automatico che contengano analisi del codice per la rilevazione di bug, code smells² e vulnerabilità di sicurezza su più di 20 linguaggi di programmazione.

SonarQube viene impostato settando dei cosiddetti "Quality Gate" che se superati segnalano il codice come buono ed utilizzabile. Gli aspetti tenuti in conto per quantificare la code quality sono:

- Bug e vulnerabilità;
- Code Smell;
- Coverage, ovvero quanto codice viene coperto dai test;
- Duplications, ovvero quante parti di codice vengono ripetute.

Il quality gate usato in azienda richiede la presenza di zero code smell e bug/vulnerabilità e la coverage richiesta è almeno del 60%.

¹<https://www.sonarqube.org/>

²serie di caratteristiche che il codice sorgente può avere e che sono generalmente riconosciute come probabili indicazioni di un difetto di programmazione.

6.3 Unit Testing e refactoring

Il primo fattore da migliorare è stata la coverage, per farlo è stato necessario scrivere test che coprissero almeno il 60% del codice scritto concentrandosi ovviamente sulle parti più importanti del programma. La parte di testing ha richiesto una vasta parte preliminare di refactoring in quanto il codice da me scritto in precedenza non era propriamente testabile.

Cosa significa avere codice "testabile"? Scrivere codice testabile significa che le sue componenti più piccole devono essere **indipendentemente verificabili**.

Il refactoring ha quindi consistito nella separazione del progetto in diversi package, separando per esempio la parte di interfacciamento con i servizi AWS dalla logica operativa del job; a loro volta all'interno dei package si è dedicata una classe ad ogni sotto-funzionalità. Nella figura 6.3 si può vedere come è stato organizzato l'Unpack Job, la stessa struttura è stata utilizzata anche per i restanti 3 job.

Allo stesso tempo i metodi delle classi sono stati riscritti in modo tale da essere il più testabili possibile, quindi:

- Cercando di evitare, per quanto possibile, di mischiare troppo parti di codice di inizializzazione e parti di codice operative;
- Individuando parti di codice in cui fosse identificabile un input e un output e dedicarvi un metodo apposito.

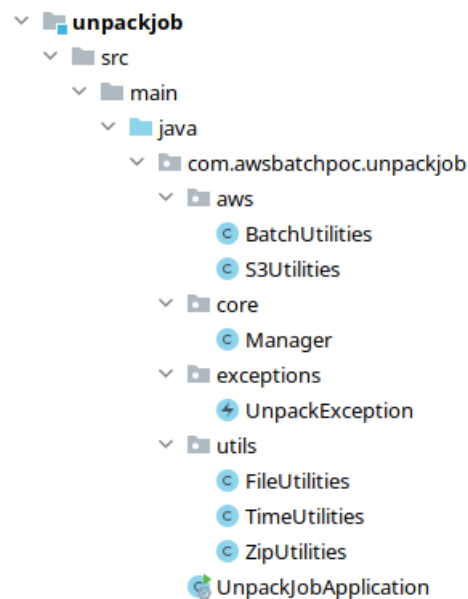


Figura 6.3: Organizzazione tipo seguita per ognuno dei 4 job

6.3.1 Unit testing con Mockito

Per creare i gli unit test ho fatto largo uso di Mockito³. Mockito è un framework di testing per Java che permette la creazione di mock object all'interno di unit test automatizzati.

Nella programmazione a oggetti i mock objects sono oggetti che simulano il comportamento di altri oggetti ma in modo controllato e controllabile. Creare oggetti di questo tipo è utile in diversi casi:

- Quando un oggetto fornisce risultati non deterministici, difficili da prevedere e quindi da testare;
- Quando si vuole testare un avvenimento raro, come un'interruzione improvvisa della rete;

³<https://site.mockito.org/>

- Quando si vuole testare un avvenimento che normalmente sarebbe molto lento.

Un mock object ha la stessa interfaccia dell'oggetto che sta andando ad imitare e per questo il resto del software non è a conoscenza della sua differenza da un oggetto normale. Il programmatore è però libero di decidere in che modo un oggetto si deve comportare in seguito a certe chiamate, specificando per esempio che all'invocazione di un metodo `method` questo ritorni sempre una certa eccezione o uno specifico risultato.

Vediamo in figura 6.4 un esempio di test per comprendere meglio il funzionamento di Mockito.

```
29      @Mock
30      AmazonS3 mockS3Client;
31
32      @InjectMocks
33      S3Utilities mockS3Utilities;
34
35      @Test
36      @DisplayName("Test object download when AmazonServiceException is threw")
37      public void downloadObjectShouldResistFailures() {
38
39          Mockito.when(mockS3Client.getObject(new GetObjectRequest( bucketName: "testBucket",
40                                                                    key: "testObjectKey")))
41              .thenThrow(new AmazonServiceException(null));
42
43          Assertions.assertNull(mockS3Utilities
44              .downloadObject( s3BucketName: "testBucket", s3ObjectKey: "testObjectKey"));
45      }
46  }
```

Figura 6.4: Testing del download di un oggetto da S3 nel caso in cui avvenga una `AmazonServiceException`

- Riga 30: creazione del mock object che imiti il client per la comunicazione con il servizio S3;

- Riga 33: il mock object creato viene iniettato nella classe `S3Utilities` andando a "sostituire" il vero oggetto di tipo `AmazonS3` di cui il costruttore avrebbe bisogno;
- Righe 39-40-41: si programma il mock object creato alla riga 30 in modo da lanciare una `AmazonServiceException` nel momento in cui viene chiamato il metodo `getObject`;
- Righe 43-44: Si verifica che il metodo `downloadObject` (che in questo caso fa uso del mock object `mockS3Client`) ritorni il risultato atteso, ovvero `null`.

I rimanenti test hanno seguito lo stesso *modus operandi* fino al raggiungimento di un accettabile 72% di coverage nell'intero progetto, come si può osservare in figura 6.5.

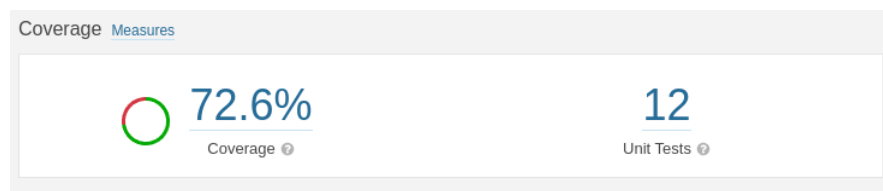


Figura 6.5: Panoramica data da sonarqube che mostra la percentuale di codice coperta dai 12 unit test realizzati

6.4 Rimozione di code smell e vulnerabilità

Quando si fa analizzare il progetto a SonarQube la piattaforma non solo ci mostra la quantità di code smell e vulnerabilità e la loro posizione ma ci dà anche suggerimenti su come sia possibile andare a risolvere questi problemi.

Ad esempio: nella figura 6.6 si può osservare come venga segnalato come possibile Security Hotspot la modalità in cui ho gestito l'apertura di un file da un archivio zip, cliccando sul tasto "See Rule" ci viene data una spiegazione molto esaustiva delle possibili conseguenze pericolose e allo stesso tempo una serie di Best Practices consigliate per evitare possibili problemi.

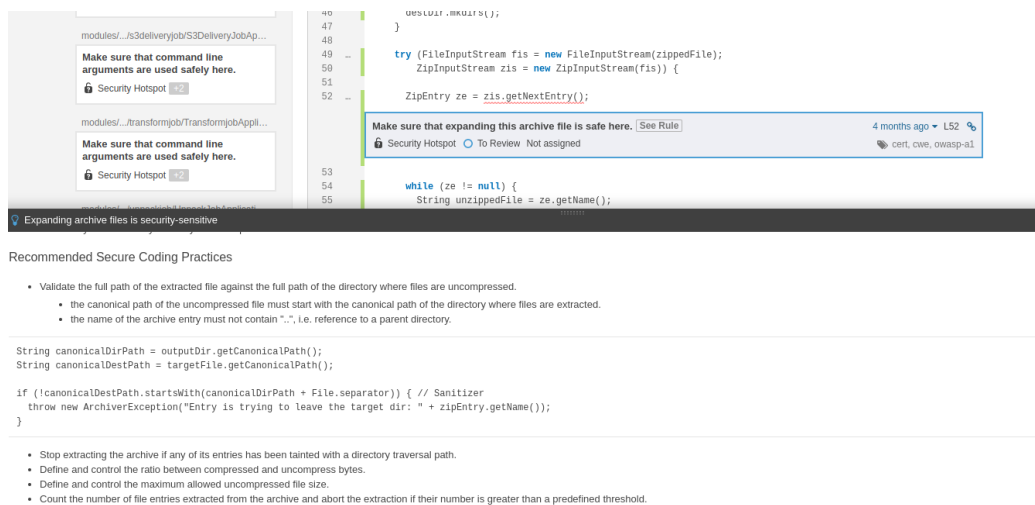


Figura 6.6: Presenza di un possibile security hotspot e code practice consigliata su come evitarlo

È buona norma risolvere il numero massimo di queste segnalazioni quando possibile e SonarQube, con le sue funzionalità, fa da ottimo assistente in questo compito.

6.5 Situazione finale

Dopo aver realizzato abbastanza unit test e corretto ogni possibile vulnerabilità la situazione è quella visibile in figura 6.7.

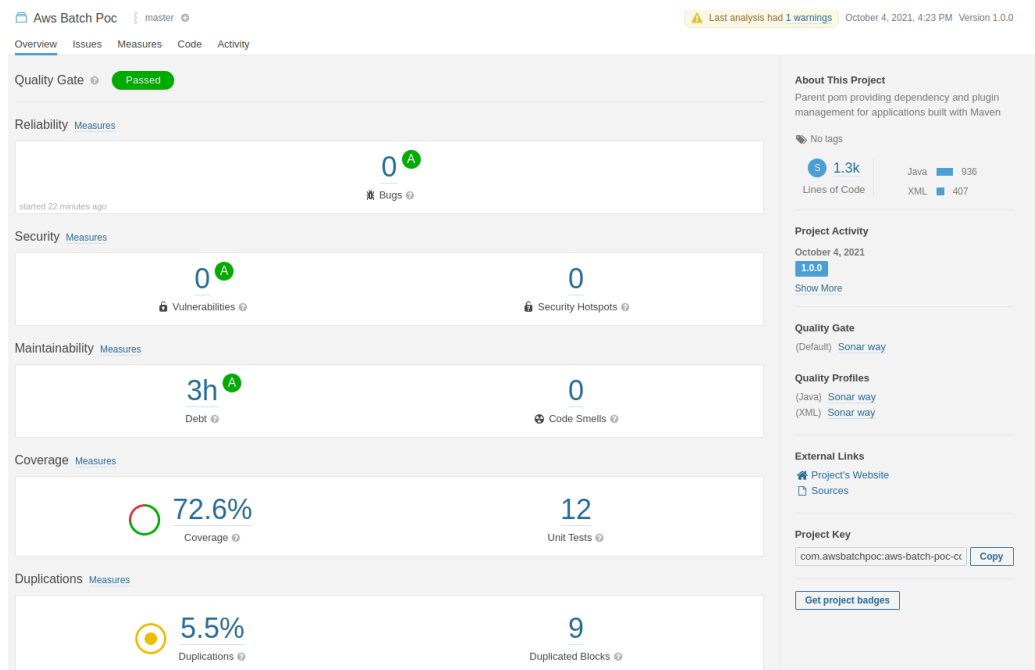


Figura 6.7: Panoramica data da SonarQube sul progetto finale

All'interno di ogni progetto aziendale tutte le pull request, ovvero tutte le modifiche al codice proposte da uno dei dipendenti devono sempre essere accompagnate da una panoramica di questo tipo, in caso contrario non verranno nemmeno prese in considerazione da parte del project manager.

Capitolo 7

Continuous Integration

L'ultima modifica effettuata al progetto è stata quella necessaria per far sì che seguisse un modello di **continuous integration**. Nell'ingegneria del software la continuous integration è una pratica che permette di automatizzare l'integrazione di modifiche al codice da parte di diverse sorgenti in un unico progetto. Seguendo questa pratica gli sviluppatori possono effettuare merge del codice molto frequenti su una repository centrale in cui il progetto viene compilato, testato ed infine eseguito.

7.1 Strumenti utilizzati

7.1.1 Bitbucket

Bitbucket¹ è un servizio realizzato da Atlassian² per progetti che usano Git come sistema di controllo versione, è simile a GitHub ma più indicato per

¹<https://bitbucket.org/>

²Compagnia che sviluppa prodotti per sviluppatori software, project management e gestione di contenuti

ambienti prettamente professionali.

7.1.2 Bamboo

Bamboo³ è un servizio, anch'esso realizzato da Atlassian, per la continuous integration e la continuous delivery (comunemente accorpati nella sigla CI/CD).

Bamboo assiste i team di sviluppatori software fornendo strumenti per:

- Building e testing automatizzato del codice;
- Aggiornamenti sull'esito delle build;
- Analisi statistica;
- Controllo degli artifact rilasciati.

7.2 Implementazione

Vediamo ora in che modo è stata implementata la continuous integration all'interno del progetto.

7.2.1 Modifica del pom e plugin Spotify

Per prima cosa è stato necessario integrare direttamente Maven e Docker, che precedentemente venivano gestiti in modo separato. Per farlo si è usato il plugin `dockerfile-maven`⁴ realizzato da Spotify.

³<https://www.atlassian.com/it/software/bamboo>

⁴<https://github.com/spotify/dockerfile-maven>

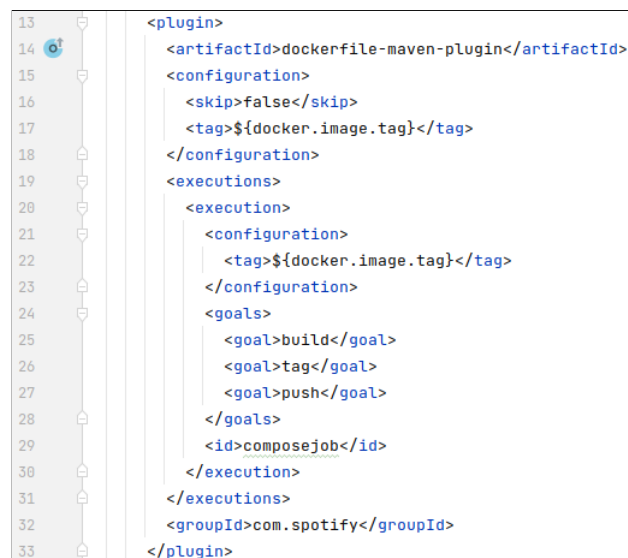
Il plugin viene utilizzato da Doxee per integrare il processo di building Docker con il processo di building Maven usando solamente i comandi di quest'ultimo. In figura 7.1 osserviamo come avvenga il setup del plugin all'interno del `pom.xml` principale.



```
10 <plugin>
11   <artifactId>spring-boot-maven-plugin</artifactId>
12   <groupId>org.springframework.boot</groupId>
13 </plugin>
14 <plugin>
15   <artifactId>dockerfile-maven-plugin</artifactId>
16   <configuration>
17     <repository>${docker.image.prefix}/${docker.image.name}</repository>
18     <skip>true</skip>
19   </configuration>
20   <groupId>com.spotify</groupId>
21   <version>${maven-docker.version}</version>
22 </plugin>
```

Figura 7.1: Setting del plugin `dockerfile-maven` nel `pom.xml` di primo livello

E in figura 7.2 possiamo invece osservare come vengano specificate queste impostazioni all'interno dei `pom` di terzo livello dei singoli job.



```
13 <plugin>
14   <artifactId>dockerfile-maven-plugin</artifactId>
15   <configuration>
16     <skip>false</skip>
17     <tag>${docker.image.tag}</tag>
18   </configuration>
19   <executions>
20     <execution>
21       <configuration>
22         <tag>${docker.image.tag}</tag>
23       </configuration>
24       <goals>
25         <goal>build</goal>
26         <goal>tag</goal>
27         <goal>push</goal>
28       </goals>
29       <id>composejob</id>
30     </execution>
31   </executions>
32   <groupId>com.spotify</groupId>
33 </plugin>
```

Figura 7.2: Setting del plugin `dockerfile-maven` nei `pom.xml` di terzo livello

Nel pom di primo livello si specifica la repository ECR per la docker image utilizzata per avviare il job, composta dal valore assegnato per la proprietà `docker.image.prefix` e `docker.image.name`, settate nella sezione properties e visibili nella figura 7.3. Nei pom di terzo livello invece si particularizzano ulteriormente le impostazioni andando a sfruttare la struttura intrinsecamente gerarchica dei progetti Maven impostando il tag per l'immagine (ogni job ha un tag diverso).

```
100 <properties>
101   <java.version>11</java.version>
102
103   <!-- Docker -->
104   <docker.image.name>aws-batch-poc-002</docker.image.name>
105   <docker.image.prefix>034135346153.dkr.ecr.us-east-2.amazonaws.com</docker.image.prefix>
```

Figura 7.3: Sezione properties nel pom di primo livello che specifica l'url della repository di destinazione

Una volta completato il setting basta lanciare uno dei comandi principali usati con un progetto Maven, ovvero `mvn clean package` ed automaticamente, oltre ad effettuare i test e a creare un jar avviabile per il job, verrà anche effettuato il building e il tagging delle immagini Docker. Lanciando il comando `mvn dockerfile:push` invece queste ultime verranno inviate al repository ECR.

7.2.2 Uso di Bamboo

Vediamo ora come far sì che il workflow spiegato alla fine della precedente sezione si adatti ad un modello di continuous integration.

Usando Bamboo è possibile definire una pipeline di comandi, script e altre operazioni che permettono in tutto e per tutto di “copiare” ciò che lo sviluppatore faceva precedentemente a mano sulla sua macchina in locale. La pipeline realizzata per il progetto è visibile in figura 7.4.



Figura 7.4: Pipeline bamboo realizzata per il progetto

In ognuna delle sezioni vengono effettuate operazioni che precedentemente venivano fatte a mano e in locale; all’interno di ogni sezione viene sempre fatto un checkout preliminare sulla repository bitbucket per usare sempre il codice più aggiornato. La pipeline è impostata per avviarsi ogni qual volta viene rilevato un nuovo push sulla repository, quando la pipeline termina Bamboo fornirà informazioni dettagliate sull’esito, sia esso stato positivo o negativo.

Andiamo ora a vedere più nel dettaglio le tre fasi della pipeline.

7.2.2.1 Initialize

In questa sezione si prepara la build andando a crearne una nuova versione e iniettandola all’interno di un file che verrà usato nelle fasi successive. Si può osservare più nel particolare in figura 7.5.

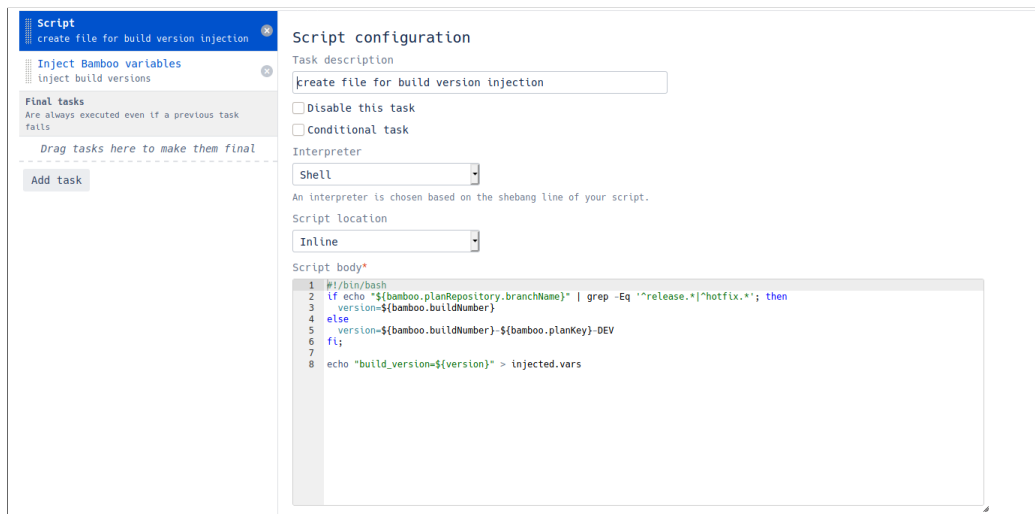


Figura 7.5: Dettaglio della fase Initialize della pipeline

7.2.2.2 Build

In questa sezione viene compilato il progetto ed effettuato il building delle immagini Docker.

7.2.2.3 Publish

In questa sezione viene i diversi job vengono pushati su ECR, particolare attenzione alla necessità in questa fase di ottenere un token di autorizzazione ECR mediante uno script bash visibile in figura 7.6 senza il quale la pubblicazione non sarebbe possibile.



Figura 7.6: Script che permette all'ambiente virtuale creato da Bamboo di essere autorizzato a pubblicare immagini sulla repository ECR del progetto

Capitolo 8

Considerazioni finali

Vediamo ora se, a progetto terminato, gli obiettivi prefissati sono stati raggiunti oppure no.

8.1 Studio dei costi

Per fare una stima dei costi, su consiglio dei miei responsabili, mi sono limitato a tener conto del `ComposeJob` e del `TransformJob` che sono quelli computazionalmente più impegnativi. In figura 8.1 una tabella di calcolo che ho realizzato per mostrare le stime fatte.

| Job | CPU | MEM (GiB) | TEMPO (s) | Esecuzioni/giorno | Storage aggiuntivo (GiB) |
|--------------|-----|-----------|-----------|-------------------|--------------------------|
| TransformJob | 1 | 0,253906 | 45 | 100 | 0 |
| ComposeJob | 1 | 0,429688 | 135 | 100 | 0 |

| STIMA DEI COSTI (CPU) | | |
|-----------------------|------------------------|------------|
| TransformJob | Costo per esecuzione = | \$0,000506 |
| | Costo mensile (30gg) = | \$1,52 |
| ComposeJob | Costo per esecuzione = | \$0,001518 |
| | Costo mensile (30gg) = | \$4,55 |

| STIMA DEI COSTI (Memoria) | | |
|---------------------------|------------------------|-----------------|
| TransformJob | Costo per esecuzione = | \$0,00014108 |
| | Costo mensile (30gg) = | \$0,42 |
| ComposeJob | Costo per esecuzione = | \$0,00071623619 |
| | Costo mensile (30gg) = | \$2,15 |

| STIMA DEI COSTI (Totale) | | |
|--------------------------|------------------------|---------|
| TransformJob | Costo mensile totale = | \$1,94 |
| ComposeJob | Costo mensile totale = | \$6,703 |
| | Tot = | \$8,64 |

Costo CPU per esecuzione = <numero cpu> * <costo cpu al secondo> * <tempo esecuzione>
Costo CPU mensile = <costo cpu per esecuzione> * <esecuzioni giornaliere> * <giorni nel mese (30)>

Costo memoria per esecuzione = <quantità memoria in GiB> * <costo mem al secondo> * <tempo esecuzione>
Costo memoria mensile = <costo memoria per esecuzione> * <esecuzioni giornaliere> * <giorni nel mese (30)>

N.B.: Nel caso si aggiungesse dello storage temporaneo ai 20 GiB di default va calcolato anche il suo costo

| | Fargate | Fargate Spot |
|---------------------|-----------------|-----------------|
| Per vCPU all'ora | \$0,040480 | \$0,012144 |
| per GiB all'ora | \$0,044450 | \$0,0013335 |
| per vCPU al minuto | \$0,0006747 | \$0,000202 |
| per GiB al minuto | \$0,0007408 | \$0,000022 |
| per vCPU al secondo | \$0,0000112444 | \$0,00000337333 |
| per GiB al secondo | \$0,00001234722 | \$0,00000037042 |

Figura 8.1: Tabella realizzata per mostrare le stime sui costi e le performance

Nella tabella a destra possiamo vedere i costi presi direttamente da AWS batch, il compute environment utilizzato era di tipo Fargate ma ho inserito anche Fargate Spot come misura di confronto. In alto ho indicato i valori utili per il calcolo dei costi, tempo d'esecuzione e memoria utilizzata sono stati estratti facendo una media su diversi test effettuati a partire da un file compresso di test simile a quelli che l'azienda riceve ogni giorno in produzione. In basso ho indicato quali formule sono state utilizzate per effettuare i calcoli. La stima è stata fatta supponendo che ogni giorno l'azienda debba avviare la catena di job batch 100 volte per un mese intero.

Il costo finale calcolato, che non tiene conto di altri servizi come S3, SNS, EFS e Lambda, è risultato molto vantaggioso secondo i miei responsabili anche se bisogna tenere comunque conto di un leggero overhead nelle performance dato da:

- Il tempo maggiore che impiega un servizio on-demand di questo tipo ad avviarsi;

- Il tempo necessario per la comunicazione di rete tra la rete aziendale e il data center Amazon.

Che sono però tollerabili in situazioni di questo tipo in cui non è richiesto alcun tipo di realtime.

Nonostante non mi sia stato possibile fare un confronto più approfondito con i dati di un workflow di produzione reale sembra che l'obiettivo sia stato raggiunto: spostare questo tipo di processi su AWS utilizzando AWS Batch conviene, non solo per i costi più ridotti ma bisogna sempre ricordare quanto l'utilizzo del cloud computing semplifichi di molto la vita alle aziende.

8.2 Limitazioni nell'utilizzo di servizi di cloud computing

Perché quindi le aziende non portano tutti i loro processi su infrastrutture come AWS se è quasi sempre così conveniente? Per le aziende e le amministrazioni pubbliche le opportunità offerte dal cloud computing sono molto interessanti, aprono la strada ad una migliore organizzazione del lavoro e delle risorse rimuovendo la necessità di gestire alcuni aspetti delle proprie infrastrutture e permettono spesso di risparmiare parecchio denaro senza andare ad intaccare in modo significativo le performance. Ma se da un lato il cloud offre una serie di importanti vantaggi la migrazione dei dati dai sistemi locali, sotto il controllo diretto del proprietario e dell'azienda, a sistemi remoti (come AWS) ha importanti implicazioni giuridiche che non possono essere sottovalutate.

8.2.1 Quali leggi applicare?

Il primo problema sorge quando è necessario decidere quale legge applicare nel caso di controversie di ogni tipo. Essendo il cloud computing basato sulla presenza di server in ogni parte del mondo, non è facile capire quale sistema giuridico applicare e a quali normative fare riferimento; nel momento in cui si decidono di usare questo tipo di servizi è molto importante leggere attentamente i contratti che ci vengono sottoposti per sapere a cosa è possibile andare incontro nel caso di controversie

8.2.2 Privacy e dati sensibili

Quando si parla di cloud computing uno degli argomenti più delicati concerne la privacy, in particolare riguardante la protezione dei dati personali dei clienti di un'azienda che si affida a questo genere di servizi. Il tema è molto complesso: quando sono presenti dati sensibili la legge prevede la necessità che tutte le parti in causa diano il consenso a gestire questi dati tramite cloud provider, pertanto in alcuni casi è possibile utilizzare servizi cloud e in altri no. Inoltre, nel caso di cloud provider extra europei, questi ultimi devono rispettare le regole del trattamento dei dati definite dal **GDPR**, ovvero il Regolamento generale sulla protezione dei dati dell'Unione Europea che tutela il diritto fondamentale alla privacy e alla protezione dei dati personali di ogni individuo. Il GDPR include rigorosi requisiti che definiscono gli standard in materia di protezione, sicurezza e conformità dei dati.