

AWS Lambda Functions



Retrieve talks info from REST APIs

Cattaneo Andrea - mat. 1040655

Locatelli Matteo - mat. 1041449

Serverless

Per lo sviluppo delle lambda functions abbiamo deciso di utilizzare il framework [serverless](#).

Serverless ci ha permesso di implementare il paradigma **Infrastructure as Code (IaC)**.

Possiamo definire lo stack del progetto AWS Lambda in un file yaml e gestire il deploy dell'applicazione da linea di comando

```
andrea@andrea-Lenovo-V330-15IKB:~/Projects/tedx_party/serverless-lambda$ serverless deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Excluding development dependencies...
Serverless: Layer commonLibs is already uploaded.
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Skip uploading commonLibs
Serverless: Uploading service serverless-lambda.zip file to S3 (2.41 KB)...
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
.....
Serverless: Stack update finished...
Service Information
service: serverless-lambda
stage: dev
region: eu-west-1
stack: serverless-lambda-dev
resources: 30
api keys:
  None
endpoints:
  POST - https://gik7hjhbgh.execute-api.eu-west-1.amazonaws.com/get-by-tag
  GET - https://gik7hjhbgh.execute-api.eu-west-1.amazonaws.com/get-watch-next
  GET - https://gik7hjhbgh.execute-api.eu-west-1.amazonaws.com/get-reviews
  POST - https://gik7hjhbgh.execute-api.eu-west-1.amazonaws.com/publish-review
functions:
  getByTag: serverless-lambda-dev-getByTag
  getWatchNext: serverless-lambda-dev-getWatchNext
  getReviews: serverless-lambda-dev-getReviews
  publishReview: serverless-lambda-dev-publishReview
layers:
  commonLibs: arn:aws:lambda:eu-west-1:884324959015:layer:commonLibs:1
Serverless: Removing old service artifacts from S3...
Serverless: Run the "serverless" command to setup monitoring, troubleshooting and testing.
```

Serverless

Nel file
serverless.yml
definiamo le diverse
lambda nella
sezione *functions*

```
functions:
  getByTag:
    handler: handler.getByTag
    memorySize: 128
    layers:
      - { Ref: CommonLibsLambdaLayer }
    events:
      - httpApi:
          method: POST
          path: /get-by-tag
```

Attraverso i layer
possiamo gestire le
dipendenze del
nostro codice

```
layers:
  commonLibs:
    path: layer
    compatibleRuntimes:
      - nodejs12.x
```

Definiamo runtime, region e
variabili d'ambiente. Nel
nostro caso la stringa di
connessione è indicata in un
file esterno che non
committiamo nel repository
Github per evitare di renderla
pubblica

```
provider:
  name: aws
  runtime: nodejs12.x
  stage: dev
  region: eu-west-1
  environment:
    DB: ${file(./environment.yml):DB}
```

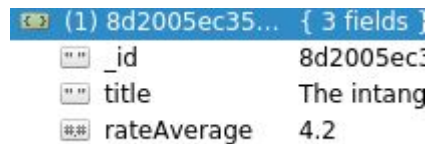
Reviews

Abbiamo realizzato una API REST che permette agli utenti di pubblicare recensioni sui talk. Ogni recensione è caratterizzata da un titolo, un contenuto, un rate (da 1 a 5) e da uno username. Le recensioni sono memorizzate in una collection chiamata *review*.

È presente una API che restituisce tutte le review di un dato video.

Se lo stesso utente carica una nuova recensione sullo stesso talk automaticamente sovrascrive la precedente.

Quando viene pubblicato una nuova recensione viene calcolato il rating medio del video. Il *rateAverage* viene quindi scritto in un attributo del documento *video*.



(1) 8d2005ec35... { 3 fields }	
_id	8d2005ec35...
title	The intang
rateAverage	4.2

Abbiamo verificato che il job periodico di aggiornamento AWS Glue non cancella i *rateAverage* esistenti.

Watch next e Get by tag

Abbiamo realizzato una API REST che permette di ottenere una lista di talk consigliati in relazione a quello attualmente in visualizzazione. I talk consigliati vengono ordinati in base al rating medio di questi.

La risposta contiene tutte le informazioni necessarie riguardo al talk: id, URL, autore, descrizione e rating. Questa scelta è stata fatta per ridurre le chiamate alle API.

La funzione GetByTag sviluppata a lezione è stata leggermente ristrutturata, implementando l'ordinamento sulla base della valutazione media del talk.

```
"_id": "5bd34fcc55d9e1267f605fa0c060d54e",
"watch_next": [
  {
    "_id": "44b0ee48c51c42aea7560ab001f08b7b",
    "rateAverage": 5,
    "..."
  },
  {
    "_id": "8d2005ec35280deb6a438dc87b225f89",
    "rateAverage": 4.2,
    "...."
  },
  {
    "_id": "4b4ea097d5b35709f90b05120100209c",
    "rateAverage": 3.6666666666666665,
    "..."
  },
  {
```

L'esperienza utente

La scelta di implementare un sistema di rating e di ordinare successivamente in base alle valutazioni medie è stata fatta nell'ottica di rendere più fruibili talk generalmente apprezzati, e penalizzare talk qualitativamente peggiori. Questo permette a nuovi utenti di visualizzare fin da subito talk di qualità, senza la necessità di dover effettuare ricerche sul web per avere informazioni sullo speaker per decidere il talk migliore.

Criticità

- È stato necessario utilizzare un account AWS “regular” perché l’account AWS EDUCATE non dispone dei permessi necessari per il funzionamento del framework serverless
- Nel momento in cui viene caricata una nuova recensione viene immediatamente aggiornato il rateAverage del video; questo potrebbe causare dei rallentamenti in presenza di molte review
- L’utilizzo del rating come metodo di ordinamento per i talk può risultare penalizzante per talk nuovi su un argomento molto trattato

Possibili evoluzioni

- Collegare al repository Github sistemi di **CI/CD** per automatizzare il deploy dello stack serverless
- Prevedere l'autenticazione degli utenti che pubblicano review
- Se sono presenti molte review su un talk aggiornare il *rateAverage* del video periodicamente e non in concomitanza alla pubblicazione di una recensione (**eventual consistency**).