RELAZIONE PROGETTO SDCC ANNO 2019/2020

Image Resources - Traccia Progetto B1

Autore:

Alessio Mazzola, Matricola: 0279323

Indice

Capitolo 1

Introduzione

Lo scopo del progetto è stato quello di sviluppare una applicazione che metta a disposizione degli utenti una suite di funzioni Serverless adibite alla modifica delle immagini. Le funzioni sviluppate sono le seguenti:

- **Resize:** Questa funzione esegue il ridimensionamento dell'immagine.
- **Black&White:** Questa funzione modifica l'immagine rendendola Bianco & Nero.
- **Brightness:** Questa funzione esegue la modificata della luminosità dell'immagine.
- **Saturation:** Questa funzione esegue la modificata della saturazione dell'immagine.

Le funzioni appena introdotte sono state realizzate seguendo il paradigma architetturale del *Serverless Computing* utilizzando il servizio *AWS Lambda*. Per richiamare le funzioni Lambda è stato utilizzato il servizio *AWS Api-Gateway* andando a creare delle apposite *Rest-API*.

Nell'applicazione si ha una distinzione tra utenti *Registrati* ed utenti *Non Registrati*; in particolare gli utenti non registrati potranno utilizzare comunque le funzioni esposte ma registrandosi verrà anche salvato uno storico delle immagini precedentemente modificate, che saranno disponibili all'utente anche in futuro per eseguire nuovamente il download.
Lo storage delle immagini[1] sono gestite attraverso il servizio *AWS S3*.

Per quanto riguarda la registrazione degli utenti e la verifica delle credenziali di accesso, è stato utilizzato il servizio *AWS Cognito*.

Infine, per lo storico delle funzioni utilizzate dagli utenti è stata creata una tabella all'interno di *DynamoDB* la quale viene aggiornata ogni volta che un utente utilizza una funzione messa a disposizione dall'applicazione.

[1] Si intende sia l'immagine originale che quella processata del sistema.

Capitolo 2

Setup dell'applicazione

L'applicazione prima di poter essere eseguita necessita della preparazione dell'ambiente di esecuzione. In particolare questo è dovuto all'utilizzo dei servizi offerti da Amazon stessa. Utilizzando infatti un account di tipo *Educate*, oltre alla *ACCESS_KEY* ed alla *SECRET_KEY* dovremmo andare a specificare anche il *SESSION_TOKEN*. Questi tre variabili sono contenute all'interno del file *ConfigS3.py*.

E' necessario inoltre aggiornare anche il contenuto delle funzioni lambda con le chiavi di sessione appena introdotte, in quanto al loro interno, una volta processata l'immagine, verrà eseguito l'upload dell'immagine in S3 andando ad utilizzare la libreria Boto3. Per l'aggiornamento delle funzioni lambda è stato creato il file *PutLambda.py* che inserisce lo zip di queste funzioni all'interno del bucket S3. Infine risulta essere necessario specificare il path[1] delle directory di upload e di download delle immagini.

Viene riportato di seguito parte del contenuto del file *ConfigS3.py*:

```
0  S3_LAMBDASTORE                    = 's3lamdafunctionstore'
1  ##stores images
2  S3_BUCKET_RESIZE                  = 's3bucketresizefunction'
3  S3_BUCKET_BeW                     = '
       s3bucketblackwhitefunction'
4  ...
5  #SESSION
6  S3_KEY                        = 'KEY'
7  S3_SECRET                     = 'SECRET'
8  SESSION_TOKEN                 = 'TOKEN'
9  LOCATION                      = 'us-east-1'
10 ####COGNITO
11 CLIENTID                     = '3tc8id07k4id807almsi6orhme'
12 IDENTITYIDPOOL               = 'us-east-1_itDQgZIS5'
```

[1] E' possibile modificare il path all'interno del file *main.py*

Capitolo 3

Serverless - AWS Lambda

In questo capitolo andiamo ad introdurre le funzioni serverless create per l'applicativo. Il vantaggio che si ha nell'utilizzo delle funzioni serverless sta nel fatto che queste consentono di eseguire codice senza dover effettuare il provisioning né gestire server. Questo significa che effettivamente il programmatore avrà l'onere di progettare esclusivamente la funzione, senza preoccuparsi dell'infrastruttura necessaria alla sua esecuzione.

Per la realizzazione delle funzioni, è stata utilizzata la libreria *Pillow*[1], la quale risulta essere adibita alla modifica delle immagini. A causa dell'utilizzo di una libreria esterna, non è stato possibile sviluppare la funzione lambda direttamente sulla pagina di AWS, in quanto non sono disponibili gli import di librerie esterne[2]. Per risolvere questo problema è stato quindi creato un *Environment* per Python 3.8, quindi è stata installata la dipendenza. Successivamente è stato creato il file *lambda_function.py*, nella quale è presente l'effettiva implementazione della funzione lambda. Tutti questi file sono dunque stati compressi ed inseriti all'interno di S3 per utilizzarli come definizione della funzione Lambda.

Nell'implementazione delle funzioni è stato tenuto conto se la richiesta fosse stata effettuata da un utente registrato o meno. A questo scopo viene quindi utilizzata la variabile *userName* andando a controllare se sia effettivamente popolata o meno, a seconda che l'utente abbia effettuato o meno l'accesso. All'interno della funzione, oltre ad essere presente il codice per la modifica effettiva dell'immagine, è presente anche del codice che andrà ad occuparsi del successivo upload all'interno del bucket di riferimento di S3. Anche in questo caso viene sfruttata la variabile *userName*, in quanto un utente registrato disporrà della propria directory personale dove vengono salvate tutte le immagini.

[1] Riferimento alla libreria: https://pypi.org/project/Pillow/
[2] Vi è supporto solo per alcune librerie come json e boto3.

3.1 AWS Api-Gateway

Per richiamare le funzioni lambda all'interno del progetto è stato utilizzato il servizio di *API Gateway*. Il suo scopo è stato quello di permettere la creazione di *Rest-API* che andranno ad eseguire, una volta richimate, le funzioni lambda a loro collegate. I parametri della funzione vengono dunque gestiti dal metodo lambda_header(...), in particolare i valori sono specificati all'interno dell'url per essere successivamente utilizzati all'interno della funzione.

Per riuscire ad eseguire il passaggio di parametri tramite URL, all'interno di *API Gateway* è stato generato un modello di mappatura personalizzato, di tipo *application/json* che riportiamo immediatamente:

```
{
    #Modello di mappatura per la funzione Resize
    "wSize": "$input.params('wSize')",
    "hSize": "$input.params('hSize')",
    "imgName": "$input.params('imgName')",
    "userName": "$input.params('userName')"
}
```

E' stato inoltre necessario modificare anche i *Parametri della richiesta query URL*, aggiungendo in questo caso gli stessi campi che vediamo all'interno della mappatura. Attraverso questa operazione stiamo dunque permettendo all'URL di accettare i valori per le variabili che abbiamo specificato.

Un esempio di URL in riferimento alla Rest API che sviluppata per la funzione Resize è il seguente:

```
url = 'https://62j41asgt0.execute-api.us-east-1.amazonaws.
    com/default/resImg?wSize='+wSize+'&hSize='+hSize+'&
    imgName='+img.filename+'&userName='+username
```

Capitolo 4

Registrazione e Login - Cognito

In questo capitolo andiamo a parlare della registrazione e dell'accesso degli utenti. Per l'implementazione all'interno del progetto è stato utilizzato il servizio *AWS Cognito*. In particolare, del servizio *Cognito*, è stata utilizzata la *Pool di Utenti* la quale fornisce opzioni di registrazione e di accesso agli utenti dell'applicativo. Le differenti opzioni fornite ai programmatori possono essere, ad esempio, la scelta del numero minimo di caratteri della *Password* o le diverse modalità di accesso[1]. Il vantaggio nell'utilizzo del servizio *Cognito* sta nel fatto che, come programmatori, non dobbiamo preoccuparci dell'implementazione del *Back-End* per i servizi di registrazione e di login. Si ha inoltre un vantaggio dovuto alla sicurezza per gli utenti in quanto, nell'ipotesi di un attacco esterno, le informazioni sarebbero protette direttamente da Amazon.

Per interfacciarci con i servizi di *Cognito* all'interno dell'applicazione viene sfruttata la libreria *Boto3*. In particolare viene creata una istanza del seguente tipo:

```
#COGNITO
cognito = boto3.client('cognito-idp',
            aws_access_key_id=configS3.S3_KEY,
            aws_secret_access_key=configS3.S3_SECRET,
            aws_session_token=configS3.SESSION_TOKEN,
            region_name = configS3.LOCATION,
)
```

Per la gestione della registrazione e dell'accesso all'interno del progetto sono state quindi implementate le seguenti funzioni[2]:

[1] L'accesso potrebbe essere permesso sia tramite NomeUtente che tramite E-mail

[2] Le funzioni sono presenti all'interno del file *FunctionCognito.py*

- **def register_user(userName,email,password, clientID, cognito):** Questa funzione si occupa della registrazione di un nuovo utente all'interno del sistema.
- **def accedi_user(clientID, email, password, cognito):** Questa funzione si occupa della verifica delle credenziali inserite dall'utente al momento del login all'interno del sistema.

La variabile *ClientID*[3] risulta essere fondamentale all'interno dei metodi in quanto permette di effettuare il collegamento tra il metodo e l'account utilizzato per l'utilizzo del servizio. Questa variabile viene generata automaticamente quando viene eseguita la creazione del Pool di Utenti. Una ulteriore variabile[4] generata è *Pool ID*, che appunto rappresenta l'ID univoco per richiamare il Pool di riferimento.

Nel progetto, per verificare che l'indirizzo e-mail inserito dall'utente sia valido, una volta effettuata la registrazione, *Cognito* andrà ad occuparsi dell'invio di una e-mail[5] contenente un link di verifica, in modo tale da accettare l'utente solamente quando la verificà sarà stata portata a termine.

Le funzioni implementate vengono richiamate rispettivamente una volta che l'utente si trova nella pagina di accesso o di registrazione. In particolare, riferendoci alla registrazione, l'utente andrà ad inserire un *Username*, la propria *Email* e la *Password*, quindi cliccando sul pulsante *Registrazione* verrà richiamato il metodo *register_user(...)*, che si occuperà di comunicare con il servizio e di andare ad inviare alla e-mail specificata dall'utente un *Link di Verifica*[6] in modo tale da accettare formalmente l'utente all'interno del sistema.

Riferendoci all'accesso dell'utente, invece, questo avviene una volta che vengono inserite le credenziali nell'apposita pagina. Nello specifico, cliccando sul pulsante *Accedi* verrà eseguita la funzione *accedi_user(..)* andando a verificare se questa vada a buon fine. Se non dovessero esserci errori nell'esecuzione del metodo allora verrà impostata la variabile globale *nomeUtente* con il valore dell'email dell'utente.

Per quanto concerne il *Logout* dell'utente dal sistema, questo viene eseguito cliccando sull'apposito pulsante presente in tutte le pagine dell'applicazione una volta eseguito il *Login*. In particolare una volta eseguito il Logout verrà inizializzata la variabile *nomeUtente* riportandola allo stato originale.

[3] E' possibile ricavare la variabile nella la sezione *Client di app* della pagina Cognito.
[4] Tutte le variabili sono definite all'interno del file *ConfigS3.py*.
[5] Per arrivare a questo risultato è stato necessario andare a definire un *Nome di Dominio*.
[6] Se la registrazione non venisse prima verificata, tentando l'accesso questo fallirebbe.

Capitolo 5

Online Storage - S3

In questo capitolo andiamo a parlare dello storage online fornito dal servizio *AWS S3* il quale risulta essere un servizio di storage di oggetti che offre scalabilità, disponibilità dei dati e sicurezza per i dati contenuti all'interno dei Bucket. Il vantaggio che si ha nell'utilizzo di S3 sta nel fatto che la sicurezza dei dati è garantita in quanto questi vengono replicati all'interno di altri server nel datacenter della medesima area geografica.

All'interno del progetto il servizio è stato utilizzato per lo storage delle immagini inserite dall'utente e per quelle processate dal sistema stesso. In particolare queste due tipologie sono state divise andando a creare due diversi *Bucket*, uno adibito all'upload delle immagini dell'utente, e l'altro utilizzato dal sistema stesso per eseguire l'upload dell'immagine modificata attraverso la funzione *Lambda*.

Per interfacciarci con il servizio S3 all'interno del progetto è stato necessario utilizzare la libreria *Boto3* messa a disposizione da Amazon e quindi andare a creare una istanza del seguente tipo:

```
0 s3 = boto3.resource(
1             's3',
2             aws_access_key_id=ACCESS_KEY_ID,
3             aws_secret_access_key=ACCESS_SECRET_KEY,
4             aws_session_token=SESSION_TOKEN,
5             config=Config(signature_version='s3v4')
6         )
```

Per la gestione dell'*Upload* e del *Download* delle immagini all'interno dei rispettivi Bucket sono state implementate le seguenti funzioni, presenti all'interno del file *FunctionS3.py*:

- **def addToBucket(imgPath, imgName, s3, BUCKET_NAME):** Questa funzione si occupa di eseguire l'upload dell'elemento specifico all'interno del Bucket S3.

- **def downloadFromBucket(imgPath, imgName, BUCKET_NAME, s3):** Questa funzione si occupa di eseguire il download dell'elemento specifico presente all'interno del Bucket S3.

Queste funzioni vengono richiamate all'interno del progetto una volta che viene richiesto dall'utente la modifica di una immagine. In particolare, quando l'utente andrà a scegliere l'immagine da modificare effettuandone l'upload nell'applicazione, verrà richiamata la funzione *addToBucket(...)* che andrà ad aggiungere l'immagine orginale all'interno del Bucket di riferimento. Successivamente verrà richiamata la funzione Lambda, la quale andrà a ad eseguire il *Download* dell'immagine inserita dall'utente all'interno del Bucket per andare a processarla e successivamente andare ad eseguire un nuovo *Upload* dell'immagine processata all'interno del Bucket. Una volta eseguito questo, all'interno dell'applicazione verrà eseguita la funzione *downloadFromBucket(...)*, la quale collegandosi al Bucket andrà a scaricare l'immagine processata, salvandola in locale all'interno del server per permettere all'utente di eseguire il *Download*. L'utilizzo di questa funzione si ha anche quando viene richiesto il download di una immagine precedentemente processata, le quali sono sempre disponibili per gli utenti registrati.

Oltre allo storage delle immagini, il servizio S3 è stato utilizzato anche per lo storage delle funzioni Lambda. E' stato infatti creato un Bucket apposito che le contenesse in modo tale che, dal pannello di controllo di *AWS Lambda*, queste venissero "collegate" andando a specificare il link di riferimento dell'archivio Zip che contenesse la specifica della funzione. Come detto in precedenza, questo passaggio è stato obbligato, in quanto sono state utilizzate librerie esterne per la creazione delle funzioni.

Capitolo 6

Online Database - DynamoDB

In questo capitolo andiamo a parlare del Database fornito dal servizio *AWS DynamoDB*. In particolare DynamoDB risulta essere un database (NoSQL) che supporta i modelli di dati di tipo chiave-valore. Uno dei vantaggi principali sta nella scalabilità del servizio, in quanto è stato espressamente progettato per questo.

All'interno del progetto questo servizio è stato utilizzato per mantenere traccia delle immagini processate per ogni utente registrato andando quindi a mantenere uno storico. Per questo scopo è stata creata la tabella *Utenti* con chiave pari all'*userName* dell'utente, e quindi la propria e-mail, mentre il valore è dato dal *Link della risorsa in S3* associata all'immagine modificata dalla funzione Lambda.

Per interfacciarci con il servizio DynamoDB all'interno del progetto è stato necessario utilizzare la libreria *Boto3* messa a disposizione da Amazon e quindi andare a creare una istanza del seguente tipo:

```
dynamodb = boto3.resource(
            'dynamodb',
            aws_access_key_id=configS3.S3_KEY,
            aws_secret_access_key=configS3.S3_SECRET,
            aws_session_token=configS3.SESSION_TOKEN,
            region_name=configS3.LOCATION,
            config=Config(signature_version='s3v4')
        )
```

Per quanto riguarda la gestione dell'inserimento e del recupero delle *entry* all'interno della tabella sono state implementate le seguenti funzioni, presenti all'interno del file *FunctionDynamo.py*:

- **def add_element_in_table(dynamodb, tableName, nomeUtente, linkS3):** Questa funzione si occupa di inserire la *entry* all'interno della tabella specificata.

- **def get_element_from_table(dynamodb, tableName, nomeUtente):** Questa funzione si occupa di recuperare la *entry* corrispondente ai valori desiderati all'interno della tabella.

La funzione *add_elemente_in_table(...)* viene utilizzata una volta che viene processata un immagine dal sistema; questa infatti viene richiamata una volta che l'utente esegue una delle quattro funzioni esposte dall'applicativo, andando quindi ad aggiungere all'interno della tabella un nuovo elemento pari a [NomeUtente - LinkS3] in modo tale da andare a mantenere uno storico per ogni utente delle immagini che sono state modificate.

Con riferimento alla funzione *get_element_from_table(...)*, questa viene richiamata quando l'utente registrato visita la pagina *Immagini Modificate*. La funzione infatti esegue una query che restituisce tutti i link di S3 inerenti allo specifico *UserName* dell'utente, i quali verranno successivamente utilizzati per andare a immettere informazioni all'interno della tabella. Il link di S3 viene inoltre utilizzato per permettere il *Download* dell'immagine direttamente dalla tabella stessa.

Capitolo 7

Online Deploy - EC2

In questo capitolo andiamo a parladere del deploy dell'applicazione. Questo viene eseguito utilizzando il servizio *EC2*. Il vantaggio principale che si ha attraverso il servizio EC2 sta nella modularità e nella vastità di macchine diverse che possono essere istanziate. In particolare, per quanto riguarda la macchina scelta, poiché le funzioni messe a disposizione dell'applicativo sono *Serverless*, non si necessita di una potenza di calcolo elevata; per questa ragione è stata scelta una macchina di tipologia *Micro*.

Per quanto riguarda il setup del progetto sarà dunque necessario stabilire un collegamento ssh tra la macchina utente e la macchina EC2, il quale viene eseguito attraverso il seguente comando:

```
0 ssh -i "chiave.pem" ec2-user@DNSPUBBLICO
```

Successivamente, sarà quindi possibile andare ad eseguire la copia dell'applicativo all'interno della directory scelta della macchina EC2 attraverso il seguente comando:

```
0 scp -i "chiave.pem" -r dirProgetto ec2-user@DNSPUBBLICO:/
    home/ec2-user/
```

Una volta quindi che il contenuto sarà stato copiato con successo, dovremmo occuparci delle dipendenze richieste per il funzionamento dell'applicazione. Sarà quindi a questo punto possibile eseguire l'applicazione all'interno della macchina EC2 istanziata.

7.1 AMI & AutoScaling

Andando a considerare lo scenario di una singola macchina, una volta che questa raggiunge il limite di utenti connessi, risulterà essere satura e dunque sarà necessario andare ad instanziare una nuova macchina al fine di poter accettare nuove richieste.

Il vantaggio principale che si ha utilizzando i gruppi di *AutoScaling* sta nel fatto che questi permettono di eseguire operazioni di *Scaling* in maniera dinamica sotto specificate regole che vengono decise al momento della creazione del gruppo. In particolare andremo ad utilizzare l'immagine AMI che contiene l'applicativo in modo tale da poter instanziare sempre macchine "uguali". Una AMI[1] risulta essere una immagine specifica della macchina virtuale. Sarà dunque nel nostro interesse andare a preparare la nostra immagine andando ad installare applicativi e dipendenze necessari all'esecuzione del progetto per poi successivamente andare a salvare l'AMI personalizzata che verrà richiamata tramite il gruppo di AutoScaling.
Quando parliamo di *scaling* è possibile definire:

- *Scale Up*: In questo caso andiamo ad aggiungere un numero *n* di nuove istanze a quelle già disponibili a causa dell'elevata richiesta degli utenti.
- *Scale Down*: In questo caso andiamo a rimuovere un numero *n* di istanze da quelle presenti in quanto non servono più per la gestione attuale degli utenti.

7.2 Load Balancer

Un *Load Balancer* è associabile ad un dispositivo posto tra l'utente e le istanze disponibili in modo tale da andare a distribuire equamente il traffico utente generato andando a sfruttare il gruppo di AutoScaling che è stato appositamente creato.

Uno dei maggiori vantaggi offerti dal Load Balancer sta nella sicurezza e nella stabilità del server, dal momento che il traffico che un server lento viene automaticamente inoltrato ad un altro server. Il Load Balancer va inoltre ad aumentare l'affidabilità dell'applicativo poiché, se uno dei server non dovesse essere raggiungibile, allora il traffico non sarà inoltrato a quel server, ma saranno gli altri server disponibili a doversene occupare fino a che il guasto non verrà risolto.

[1] Amazon Machine Image

Capitolo 8

Link e Limitazioni

8.1 Limitazioni

Durante lo sviluppo dell'applicazione sono stati riscontrate alcune limitazioni dovute all'utilizzo dell'account *Educate* per AWS. La limitazione più grande è stata l'"impossibilità di creazione di utenti IAM in quanto la creazione di account avrebbe permesso di creare account con determinati permessi, in modo da non dover modificare ogni volta le chiavi della sessione, pregiudicandone quindi l'utizzo dei servizi di AutoScaling e del LoadBalancer.

8.2 Link

Per quanto concerne i Link riportiamo di seguito la repository GitHub contenente il progetto:

- GitHub: https://github.com/GlaAndry/SitoProgCardellini