Relazione Progetto Big Data – Stefano Ziantoni 65171

Creazione di un cluster standalone su istanze EC2 con Spark, salvataggio dei risultati su S3, con descrizione dello sviluppo di una applicazione in Python per interrogare i risultati su S3 tramite Amazon Athena.

Sommario:

0: Introduzione:

Relazione sul progetto per l'esame di Big Data 2020/21. In questa relazione verrà descritto il procedimento di creazione di un cluster in Spark di tipo Standalone gestito tramite Terraform. Viene descritto l'intero procedimento di costruzione del cluster. Successivamente, viene eseguito tramite il cluster uno script in Python che si occupa di contare le occorrenze per riga di ogni parola in un file di testo di grandi dimensioni tramite l'utilizzo di un approccio map-reduce in Spark, che divide il lavoro tra le macchine del cluster. I risultati della computazione, così come i dati in input, vengono salvati sullo storage di Amazon S3.

Nella parte successiva dell'elaborato verrà descritta la creazione di una semplice interfaccia grafica in Django, tramite la quale sarà possibile interrogare i risultati della computazione sfruttando le query SQL-Like eseguibili in Amazon Athena su S3.

1: Creazione cluster con Spark e Terraform:

1.1- Creazione VPC

Il primo step da svolgere consiste nella creazione di una VPC (Virtual private Cloud) che ci permette di avviare le risorse di AWS in una rete virtuale personalizzata.

- Nella barra di ricerca della console AWS scrivere "VPC" e selezionare il primo risultato.
- Cliccare su "Launch VPC Wizard" e cliccare su Select.
- Assegnare un nome alla VPC e cliccare su "Create VPC".
- Una volta concluso, cliccare nel menù di sinistra la voce "Subnets"
- Dovrebbe comparire una subnet pubblica chiamata "Public Subnet". A questo punto cambiare il nome della subnet per renderla riconoscibile.
- Selezionare la subnet e cliccare su "Actions", e selezionare la voce "Modify auto-assign IP settings". Quindi selezionare "Auto-assign IPv4". In questo modo sarà possibile comunicare con le istanze tramite SSH perché AWS assegnerà loro IPv4 pubblici.
- Abbiamo quindi creato la subnet all'interno della quale creeremo le istanze.

1.2 Creazione istanza t2.micro con Spark

Creazione istanza principale:

Dalla console AWS selezionare la voce "EC2". Quindi selezionare la voce "Istances" dal menù di sinistra. Poi selezionare "Launch Instances"

- Come AMI (Amazon Machine Image) selezionare Ubuntu 18.04 a 64-bit.
- Selezionare il tipo di istanza, nel nostro caso t2.micro
- Cliccare su "Next: Configure Instance Details"
- Impostare a 1 il numero di istanze, si andrà a creare solo l'istanza di base e le altre verranno create tramite l'AMI di quest'ultima.
- Alla voce "Network" scegliere la VPC che è stata creata nel passaggio precedente.
- Cliccare su "Next: Add Storage"
- Impostare 30 Giga di storage, il massimo consentito
- Cliccare su "Next: Add Tags" e su "Next: Configure Security Group"
- Creare un nuovo gruppo di sicurezza tramite "Create a new security group" e impostare il nome nel campo "Security group name"
- Cliccare su "Review and Launch" e poi "Launch"
- Una volta cliccato vi verrà chiesto di creare una nuova coppia di chiavi. Selezionare la voce "Create a new key pair" e impostare un nome in "Key pair name", poi cliccate su "Download Key Pair" che scaricherà la chiave privata con estensione ".pem". Tramite la chiave privata salvata in locale sarà possibile connettersi alle istanze via SSH.

- Cliccare quindi su "Launch Instances" per confermare.
- Una volta creata l'istanza tornare alla sezione "Instances" e dare un nome all'istanza cliccando sulla matita. Nel nostro caso sarà "namenode".

Una volta concluso questo processo, sarà necessario modificare il Gruppo di Sicurezza.
Dal menù a sinistra di AWS selezionare "Security Group" in Network & Security.
Una volta caricato il menù dei gruppi di sicurezza, selezionare il gruppo di sicurezza personalizzato creato in precedenza.
Nel menù che appare in basso selezionare "Inbound rules", cliccare su "Edit inbound rules". Quindi cliccare su "Add rule".
Scegliere come opzione "All Traffic" e come Source selezionare "Custom", quindi scrivere il CIDR scelto per la subnet (dovrebbe essere sempre 10.0.0.0/24).
Quindi salvare la regola con "Save rules".

Connessione all'istanza:

Nel menù "Istances" cliccare sull'istanza appena creata con il testo destro. Quindi cliccare su "Istance State" o "Stato dell'istanza" nel menù, e quindi su "Start" o "Inizio".
Una volta che l'istanza è stata avviata, cliccare nuovamente con il destro sull'istanza e quindi su "Connect" o "Collegarsi".
Copiare la striga per la connessione SSH.

Aprire quindi una shell Linux nella cartella dove è stata scaricata la chiave.
Nella shell scrivere "sudo" e successivamente incollare la stringa di connessione. (Nella forma ssh -i chiave.pem ubuntu@INDIRIZZO DNS PUBBLICO DELL'ISTANZA).
Una volta connessi all'istanza, aprire un'altra shell nella stessa cartella e digitare il comando:

"scp -i 'chiave.pem' chiave.pem ubuntu@ INDIRIZZO DNS PUBBLICO DELL'ISTANZA*:/home/ubuntu/.ssh"*

Chiudere la shell una volta inviata la chiave e tornare sulla prima shell, digitare quindi:

"chmod 400 /home/ubuntu/.ssh/chiave.pem"

Ora, nella shell connessa all'istanza namenode scrivere:

"sudo nano /etc/hosts"

E scrivere nel file:

IP PRIVATO DI NAMENODE namenode
IP PRIVATO DI NAMENODE datanode1

Salvare, quindi scrivere:

"nano /home/ubuntu/.ssh/config"

E scrivere nel file:

```
Host namenode
HostName namenode
User ubuntu
IdentityFile /home/ubuntu/.ssh/[NOME CHIAVE].pem
Host datanode1
HostName namenode
User ubuntu
IdentityFile /home/ubuntu/.ssh/[NOME CHIAVE].pem
```

Set-up Java e Spark

Aggiornare la macchina e istallare Java e Spark tramite i comandi:

"sudo apt-get update && sudo apt-get dist-upgrade"

"sudo apt-get install openjdk-8-jdk"

"wget https://archive.apache.org/dist/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz"

"tar -xvzf spark-2.4.4-bin-hadoop2.7.tgz"

"sudo mv ./spark-2.4.4-bin-hadoop2.7 /home/ubuntu/spark"

"rm spark-2.4.4-bin-hadoop2.7.tgz"

"sudo cp spark/conf/spark-env.sh.template spark/conf/spark-env.sh"

Quindi modificare le variabili d'ambiente:

"sudo nano /etc/environment"

E scrivere nel file:

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:"
JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
```

Quindi:

"source /etc/environment"
"nano /home/ubuntu/.profile"

E scrivere nel file:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$PATH:$JAVA_HOME/bin
```

E infine:

```
"source /home/ubuntu/.profile"
```

Successivamente Andiamo a concludere la configurazione di Spark:

```
"sudo nano spark/conf/spark-env.sh"
```

E scrivere all'interno del file:

```
export SPARK_MASTER_HOST=namenode
export PYSPARK_PYTHON="/usr/bin/python3"
```

Andiamo quindi a creare il file slaves che ci sarvirà per far partire tutti gli slaves del cluster con un unico script.

```
"nano spark/conf/slaves"
```

Salvare il file senza scrivere nulla. Nel caso di questo progetto, il namenode è solo master e tutti gli slave per spark sono creati tramite Terraform. Anche questo ultimo file sarà aggiornato in automatico tramite Terraform.

L'ultima cosa da fare per concludere la configurazione dell'istanza è istallare "pandas". Avremo bisogno di questa libreria per la fase di testing.

Eseguire i seguenti comandi:

```
"sudo apt install python3-pip"
"python3 -m pip install pandas"
```

1.3 Copiare l'AMI

A questo punto, conclusa la configurazione dell'istanza namenode, andremo a creare una copia AMI, uno snapshot dell'istanza che permetterà a Terraform di ricreare istanze con gli stessi settaggi di namenode in automatico. Saranno le istanze utilizzate nel cluster.

Per farlo, dal menù di EC2 cliccare con il tasto destro sull'istanza namenode, quindi "Image" e "Create Image". Scegliere un nome per l'immagine e cliccare su "Create Image".

Una volta creata l'immagine, quest'ultima sarà accessibile dal menù sulla sinistra alla voce "AMI" sotto "Images".

1.4 AWS CLI

Installiamo la CLI AWS in modo che poi possa essere utilizzata da Terraform per accedere alle nostre risorse:

"curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip""

"unzip awscliv2.zip"

"sudo ./aws/install"

"rm awscliv2.zip"

Poi aprire la console AWS. Cliccare sul proprio nome utente in alto a destra e scegliere "Le mie credenziali di sicurezza" Aprire il tab "Chiavi di accesso" e selezionare "Crea nuova chiave di accesso". Si aprirà una finestra di dialogo da cui si potranno scaricare le chiavi cliccando "Scarica file di chiavi".

Una volta scaricato il file scrivere sulla shell:

"aws configure"

E inserire i dati uno alla volta presenti nel file scaricato.
Come regione impostare quella della propria istanza, come formato di output scrivere "json".

1.5 Configurazione Terraform

A questo punto possiamo configurare Terraform. Infatti, non sarà necessario che sia presente nella copia dell'AMI perché verrà usato solo su namenode.
Eseguire il comando:

"wget https://releases.hashicorp.com/terraform/0.12.24/terraform_0.12.24_linux_amd64.zip"

Installare unzip per estrarre il file zip scaricato

"sudo apt install unzip"

Quindi estrarre il contenuto del file zip e rimuoverlo:

"unzip terraform_0.12.24_linux_amd64.zip"

"rm terraform_0.12.24_linux_amd64.zip"

Quindi creiamo la cartella per Terraform e mettiamo al suo interno la cartella estratta:

"mkdir Terraform"

"mv terraform Terraform/"

Ora sarà necessario aggiornare le variabili d'ambiente, digitare:

"sudo nano /etc/environment"

E scrivere alla fine della stringa PATH aggiunta prima:

":/home/ubuntu/Terraform"

Quindi:

"source /etc/environment"

A questo punto sarà necessario creare il file di configurazione di Terraform, chiamato "main.tf"

Digitare:

"nano Terraform/main.tf"

E scrivere all'interno del file:

```
provider "aws" {
  profile = "default"
  region = "[REGION]"
}
resource "aws_instance" "testInstances" {
   ami = "[ID AMI]"
   instance_type = "t2.micro"
   subnet_id = "[ID SUBNET]"
   vpc_security_group_ids = [
      "[ID SECURITY GROUP]",
   ]
   count = [NUMBER OF INSTANCES TO CREATE]
}
resource "null_resource" "testInstances" {
   provisioner "local-exec" {
      command = join("_", aws_instance.testInstances.*.private_ip)
       interpreter = ["bash", "/home/ubuntu/Setup.sh", "[KEY NAME]", "[INDEX START]"]
   }
   provisioner "local-exec" {
      when = destroy
      command = [NUMBER OF INSTANCES TO CREATE]
       interpreter = ["bash", "/home/ubuntu/Clear.sh", "[INDEX START]"]
      on_failure = continue
   }}
```

Come si nota, in questo file sono presenti una serie di opzioni da specificare:

- **[REGION]**: regione utilizzata (ad esempio us-east-1)
- **[ID AMI]**: ID dell'AMI che si può trovare nell'AMI su AWS cliccandoci sopra e aprendo il menù in basso.
- **[SUBNET ID]**: L'ID della VPC create inizialmente, che si trova nel menù VPC cliccando sulla VPC Creata e aprendo il menù in basso.
- **[ID SECURITY GROUP]**: ID del Gruppo di sicurezza creato in precedenza, che si trova nel campo Security Group del menù di AWS, cliccando sul security group e aprendo il menù in basso.
- **[NUMBER OF INSTANCES TO CREATE]**: Il numero di istanze che vogliamo utilizzare nel cluster.
- **[KEY NAME]**: il nome della chiave scaricata in locale
- **[INDEX START]**: l'indice di partenza da cui nominare i nuovi nodi, ad esempio mettendo 2 e creando un cluster di tre nodi, questi si chiameranno datanode2, datanode3, datanode4.

Inoltre, nel file, sono presenti anche due chiamate a script esterni, uno di Setup del cluster, che imposta le istanze appena create in automatico, e uno di pulizia del cluster che riporta all'impostazione iniziale precedente alla creazione delle istanze tramite Terraform.

Creare il primo file con il comando:

"nano Setup.sh"

E scrivere all'interno del file:

```
#!/bin/bash

cat /etc/hosts > /home/ubuntu/.tmpHosts

cat /home/ubuntu/.ssh/config > /home/ubuntu/.tmpSSHConfig

index=$2

IFS='_' read -ra IPs <<<$3

for i in ${IPs[@]}; do

            awk -v ip="$i" -v idx="$index" '!x{x=sub(/^$/,ip" datanode"idx"\n")}1' /etc/hosts > _tmp &&
            sudo mv _tmp /etc/hosts

            echo -e "Host datanode${index}\nHostName datanode${index}\nUser ubuntu\nIdentityFile
            /home/ubuntu/.ssh/${1}.pem" >> /home/ubuntu/.ssh/config

            echo "datanode${index}" | sudo tee -a /home/ubuntu/spark/conf/slaves

            index=$((index + 1))

done
```

Lo script si occupa di configurare la connessione ssh e la configurazione di Spark per ogni datanode.

Poi creiamo il secondo file:

"nano Clear.sh"

E scriviamo all'interno del file:

```
#!/bin/bash

n_datanodes=$2

END=$((n_datanodes+2))

for ((i=$1;i<END;i++)); do

     ssh-keygen -f "/home/ubuntu/.ssh/known_hosts" -R "datanode"$i

done

sudo echo " " > /home/ubuntu/spark/conf/slaves

sudo mv /home/ubuntu/.tmpHosts /etc/hosts

sudo mv /home/ubuntu/.tmpSSHConfig /home/ubuntu/.ssh/config

sudo rm -r /tmp/*
```

Salviamo e diamo i permessi ai file appena creati:

"chmod 777 Setup.sh"

"chmod 777 Clear.sh"

Infine, sarà necessario creare un terzo script, attraverso il quale sarà possibile aggiornare la configurazione interna dei datanodes una volta creati:

"nano settingNewNodes.sh"

E scrivere all'interno del file:

```
 #!/bin/bash

n_datanodes=$2

END=$((n_datanodes+2))

for ((i=$1;i<END;i++)); do

    cat /etc/hosts | ssh -oStrictHostKeyChecking=no datanode$i "sudo sh -c 'cat >/etc/hosts'"

    cat /home/ubuntu/.ssh/config | ssh -oStrictHostKeyChecking=no datanode$i "sudo sh -c 'cat
>/home/ubuntu/.ssh/config'"

done
```

Salvare il file e dare i permessi con il comando:

"chmod 777 settingNewNodes.sh"

Per eseguire questo script bisognerà scrivere:

"bash settingNewNodes.sh ***[INDEX START] [NUMBER OF INSTANCES CREATED]"***

1.6 Creazione Cluster

A questo punto avremo tutto impostato per creare il cluster.
Dopo aver popolato il file main.tf con le impostazioni desiderate, spostarsi nella cartella di /Terraform e digitare:

"terraform init"
"terrafrom apply"

Terraform provvederà a creare le istanze secondo le informazioni specificate nel file main.tf. Quando terraform ha concluso la creazione delle istanze, tornare nella home e eseguire:

"*bash settingNewNodes.sh* ***[INDEX START] [NUMBER OF INSTANCES CREATED]***"

Nel caso in cui volessimo eliminare il cluster creato, sarà possibile eseguire "*terraform destroy"* nella cartella di terraform e riportare così il tutto alla impostazione precedente la creazione del cluster.

2**: Testing, risultati e salvataggio su S3**

2.1 Introduzione test

Nel caso di questo progetto, il cluster utilizzato è composto da 10 nodi. Un nodo master e 9 nodi slave. Si è scelto di non usare il master anche come slave perché creava problemi nel submit a Spark. Quindi, una volta specificato in Terraform 9 come numero di istanze da creare e 2 come indice di partenza, è stato creato il cluster tramite gli script Terraform esposti in precedenza.

Il test svolto su questo cluster utilizza un file di testo come input, e tramite un algoritmo di map-reduce effettua il conteggio delle occorrenze per riga di ogni parola, producendo come risultato un array di coppie (parola, N° di righe in cui compare).

Il file di test in input è salvato su S3, e anche il risultato della computazione sarà salvato come CSV su S3.

2.2 S3:

Per utilizzare lo storage di Amazon S3 i passi sono semplici e veloci.

- Nella barra di ricerca della console AWS digitare S3 e selezionare il primo risultato.
- Cliccare sul testo "Create Bucket"
- Dare un nome al bucket e cliccare "Create Bucket" in fondo alla pagina.
- Una volta creato il bucket, cliccarci sopra e aprirlo
- Successivamente, cliccare su "Upload" e seguire la procedura di caricamento del file di testo selezionando il file "dump.txt" nel repository Gitub dopo averlo scaricato.

Una volta caricato il file di testo su S3, sarà possibile utilizzarlo in Spark.

Nel caso in cui il file dump.txt fosse troppo grande, è possibile splittarlo con il comando:

"split dump.txt -b [x]m –additional-suffix=.txt"

Dove [x] indica I megabyte di dimensione di ciascun file splittato.

2.3 Submit al cluster

Lo script che verrà utilizzato è lo script "example.py" presente nel repository di GitHub.

```
import […]

if __name__ == "__main__":
       data = []
       sc = SparkContext()
       file_ = sc.textFile("s3a://[MY BUCKET]/dump.txt")

      counts = file_.map(lambda line: [(i, 1) for i in set(line.split(" "))]).flatMap(lambda
x: x).reduceByKey(lambda x, y: x + y).collect()

       for i in range(0, 1000):
           print(counts[i])

       #df = pandas.DataFrame(counts, columns=['word', 'count'])

       #btw = df.to_csv(index = False).encode()

       #fs = s3fs.S3FileSystem(key=[AWS ACCESS KEY ID]', secret=[AWS SECRET KEY ID])

       #with fs.open('s3://[RESULTS BUCKET NAME]/file_test.csv', 'wb') as f:
       #       f.write(btw)

       sc.stop()
```

Nello script viene utilizzato sc.textFile per leggere il file da S3. Per poter utilizzare "s3a" dovremo includere questo framework nel comando di submit che verrà esposto a breve, e inoltre avremo bisogno di esportare le nostre credenziali AWS quindi sarà necessario digitare e inviare i comandi:

*export AWS_ACCESS_KEY_ID=**[ID AWS]***
*export AWS_SECRET_ACCESS_KEY=**[SECRET KEY AWS]***

Entrambe le credenziali si possono trovare nel file scaricato in precedenza nella configurazione della CLI.
Una volta fatto questo sarà possibile fare il submit dello script al cluster in modalità Standalone.
Per farlo, prima di tutto, bisognerà avviare Spark con i comandi da eseguire nella home:

"./spark/sbin/start-master.sh"
"./spark/sbin/start-slaves.sh"

Una volta eseguiti i comandi, controllare che tutti i nodi slaves sono stati avviati correttamente.
Per farlo, prima di tutto, torniamo nella console AWS e clicchiamo su "Security Group".
Selezioniamo il nostro gruppo di sicurezza, e nel menù in basso selezioniamo "Inboud" e aggiungere la regola "All Traffic – my IP".
Successivamente sarà possibile visualizzare il cluster avviato nel sito web:

http://INDIRIZZO_DNS_PUBBLICO_ NAMENODE:8080

Se il cluster risulta avviato correttamente, è possibile fare il submit dello script con il comando:

*./spark/bin/spark-submit --packages com.amazonaws:aws-java-sdk:1.7.4,org.apache.hadoop:hadoop-aws:2.7.7 --master spark://namenode:7077 --**executor-memory** 1G --**total-executor-cores** 9 example.py*

Notiamo come nel commando vengano inclusi i pacchetti necessari per utilizzare s3a, e inoltre sono presenti le opzioni "**totale-executors-cores**" che ci permette di scegliere quanti core utilizzare (nel nostro caso ogni macchina ha un solo core quindi questo numero coincide con il numero di macchine tra le quali spartire il lavoro), e l'opzione **"executor-memory",** che invece ci permette di specificare quanta memoria assegnare a ogni core.

Nello script la parte commentata dopo la stampa del risultato di counts è la parte di codice utilizzata per scrivere i risultati su S3 una volta conclusa la computazione (ovviamente vengono scritti una volta sola e non in tutti i test). Questa fase trasforma il risultato in Dataframe e successivamente utilizza la libreria S3fs per scrivere un file CSV su S3. Sarà necessario fornire al filesystem le credenziali di accesso alla CLI AWS.
Per poter salvare i risultati sarà necessario creare un bucket su S3 per contenerli, nel caso di questo progetto è stato creato il bucket "**resultsziantoni**".

2.4 Risultati

I test svolti vedono l'utilizzo di diverse configurazioni di cluster, con più o meno nodi e più o meno memoria. Sono stati utilizzati tre file, uno da 500 MB, uno da 1 GB entrambi porzioni del terzo file da 1.5 GB, i risultati sono:

Application ID	Name	Cores	Memory per Executor	Submitted Time	Duration	Size
app-20210601171042-0000	example.py	9	1024.0 MB	01/06/2021 17:10	29 s	500MB
app-20210601171133-0001	example.py	5	1024.0 MB	01/06/2021 17:11	36 s	500MB
app-20210601171237-0002	example.py	3	1024.0 MB	01/06/2021 17:12	45 s	500MB
app-20210601171408-0003	example.py	2	1024.0 MB	01/06/2021 17:14	60 s	500MB
app-20210601171607-0004	example.py	1	1024.0 MB	01/06/2021 17:16	1.7 min	500MB
app-20210601172146-0000	example.py	9	500.0 MB	01/06/2021 17:21	49 s	1GB
app-20210601172321-0001	example.py	5	500.0 MB	01/06/2021 17:23	1 min	1GB
app-20210601172508-0002	example.py	3	500.0 MB	01/06/2021 17:25	1.3 min	1GB
app-20210601172653-0003	example.py	2	500.0 MB	01/06/2021 17:26	1.8 min	1GB
app-20210601172915-0004	example.py	1	500.0 MB	01/06/2021 17:29	3.4 min	1GB
app-20210601161732-0000	example.py	9	1024.0 MB	01/06/2021 16:17	42 s	1GB
app-20210601161901-0001	example.py	5	1024.0 MB	01/06/2021 16:19	59 s	1GB
app-20210601162029-0002	example.py	3	1024.0 MB	01/06/2021 16:20	1.3 min	1GB
app-20210601162215-0003	example.py	2	1024.0 MB	01/06/2021 16:22	1.8 min	1GB
app-20210601162434-0004	example.py	1	1024.0 MB	01/06/2021 16:24	3.4 min	1GB
app-20210601183914-0005	example.py	9	500.0 MB	01/06/2021 18:39	50 s	1.35GB
app-20210601184031-0006	example.py	5	500.0 MB	01/06/2021 18:40	1.2 min	1.35GB
app-20210601184208-0007	example.py	3	500.0 MB	01/06/2021 18:42	1.7 min	1.35GB
app-20210601184418-0008	example.py	2	500.0 MB	01/06/2021 18:44	2.4 min	1.35GB
app-20210601184708-0009	example.py	1	500.0 MB	01/06/2021 18:47	4.5 min	1.35GB
app-20210601182601-0000	example.py	9	1024.0 MB	01/06/2021 18:26	49 s	1.35GB
app-20210601182719-0001	example.py	5	1024.0 MB	01/06/2021 18:27	1.2 min	1.35GB
app-20210601182857-0002	example.py	3	1024.0 MB	01/06/2021 18:28	1.7 min	1.35GB
app-20210601183107-0003	example.py	2	1024.0 MB	01/06/2021 18:31	2.5 min	1.35GB
app-20210601183406-0004	example.py	1	1024.0 MB	01/06/2021 18:34	4.6 min	1.35GB

3: Interfaccia in Django per query con AWS Athena su S3

A questo punto verrà descritta la creazione dell'ambiente Django e verrà descritta l'interfaccia grafica e la gestione delle query da Python ad Amazon Athena.

3.1 Configurazione Django

L'ambiente Django e l'applicazione sono stati sviluppati in locale in un ambiente Python 3.8, ovviamente la stessa configurazione può essere riprodotta su un istanza EC2 con Python come quelle utilizzate finora. La scelta di sviluppare il tutto in locale è stata fatta per contenere i costi del Free Tier di AWS.
In generale, verranno comunque trattate entrambe le configurazioni.

Prima di tutto, bisognerà controllare la presenza di Python e Pip.

Nell'istanza EC2 dovremmo averli già configurati in precedenza. In locale è necessario eseguire i seguenti comandi:

```
"sudo apt-get update && sudo apt-get -y upgrade"
"sudo apt-get install python3"
"sudo apt-get install -y python3-pip"
```

Lo step successive è quello di istallare virtualenv:

```
"pip3 install virtualenv"
```

Django in locale e su EC2:

Django verrà istallato tramite pip3:

Creiamo da directory per contenere l'applicazione Django:

```
"mkdir django-apps"
"cd django-apps"
```

E all'intero della cartella appena creata, creiamo un ambiente virtuale, chiamato "env":

```
"virtualenv env"
```

E attiviamo l'ambiente con il comando:

```
". env/bin/activate"
```

Possiamo disattivare l'ambiente quando lo desideriamo con `". env/bin/deactivate"`

Si noterà che l'ambiente è stato avviato dalla presenza nella linea di comando del tag (env) all'inizio.

Una volta attivato l'ambiente, il passo successivo sarà quello di istallare Django:

```
"pip install django"
```

E una volta istallato verifichiamo la versione:

```
"django-admin --version"
```

Ora sarà possibile **creare il progetto,** creiamo un progetto Django con il nome "testsite":

```
"django-admin startproject testsite"
```

Spostiamoci nella cartella testsite e controlliamo cosa c'è dentro:

"cd testsite"

"ls"

Output: [*manage.py testsite*]

"cd testsite"

"ls"

Output: [*__init__.py settings.py urls.py wsgi.py*]

- *__init__.py* funge da punto di ingresso per il progetto Python.
- *settings.py* descrive la configurazione dell'installazione di Django e fa sapere a Django quali impostazioni sono disponibili.
- *urls.py* contiene un urlpattern che mappa gli URL alle viste nel file views.
- *wsgi.py* contiene la configurazione per l'interfaccia del gateway del server Web. La Web Server Gateway Interface (WSGI) è lo standard della piattaforma Python per la distribuzione di server Web e applicazioni.

L'ultimo passo di configurazione consiste nell'avviare il server locale e visualizare il sito web di esempio di Django. Il comando da utilizzare è *runserver*.

Prima di avviare il sito, sarà necessario aggiungere l'indirizzo IP del server all'elenco degli ALLOWED_HOST. Elenco che possiamo trovare nel file *setting.py*.

Partendo dalla cartella */django-apps*:

"nano testsite/testsite/setting.py"

E nel file cercare ALLOWED_HOST:

```
"""
Django settings for testsite project.


Generated by 'django-admin startproject' using Django 2.0.

...
"""

...
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True


# Edit the line below with your server IP address
ALLOWED_HOSTS = ['your-ip']

...
```

A questo punto, **la prima differenza di configurazione:**

Nel caso in cui la configurazione sia in locale (il nostro caso), al posto di 'your-ip' bisognerà inserire:

ALLOWED_HOST = ['*127.0.0.1*'] o ALLOWED_HOST = ['*']

Nel caso in cui invece si stia configurando l'applicazione sulla macchina EC2 bisognerà inserire:

ALLOWED_HOST = ['EC2_DNS_NAME']

E quindi bisognerà inserire l'indirizzo IP o nome DNS ad ALLOWED_HOST in settings.py.

A questo punto, tornando alla configurazione locale, assicuriamoci di essere tornati nella cartella che contiene *manage.py*

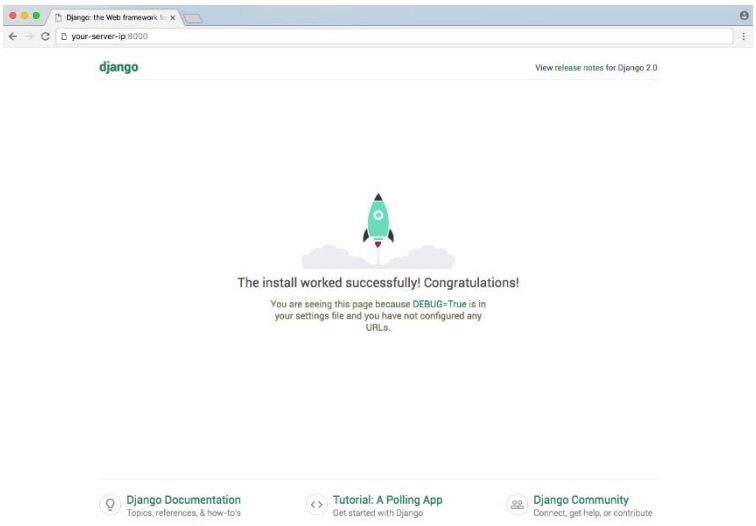
```
"cd ~/django-apps/testsite/"
```

E avviamo il sito web:

```
"python manage.py migrate"
```

```
"python manage.py runserver 127.0.0.1:8000"
```

A questo punto sarà possibile visualizzare il sito all'indirizzo: http://127.0.0.1:8000/



Invece per avviare il sito sulla macchina EC2, sulla quale ovviamente come in locale dovremo aver attivato anche l'ambiente virtuale, il comando sarà:

```
"python manage.py runserver 0.0.0.0:8000"
```

Sarà possibile visualizzare il sito all'indirizzo: : http://PUBLIC_DNS_ADDRESS:8000/

Quando però l'istanza verrà spenta, sarà necessario al prossimo avvio rifare i passaggi precedenti di configurazione aggiornando gli indirizzi DNS della macchina.

3.2 Amazon Athena

Amazon Athena è un servizio fornito da Amazon AWS per analizzare dati memorizzati su Amazon S3 tramite query interattive che rispettano lo standard SQL, è un servizio serverless che si paga solo

in base al tempo di query. Per usare Athena sarà necessario accedere al portale si AWS, creare una tabella specificando una fonte di dati in S3.

Nello script di test che è stato utilizzato era presente un pezzo di codice che si occupava di salvare i dati su un bucket S3 precedentemente creato, in formato CSV.

```
import […]

if __name__ == "__main__":
        data = []
        sc = SparkContext()
        file_ = sc.textFile("s3a://[MY BUCKET]/dump.txt")

        counts = file_.map(lambda line: [(i, 1) for i in set(line.split("
"))]).flatMap(lambda x: x).reduceByKey(lambda x, y: x + y).collect()

        for i in range(0, 1000):
            print(counts[i])

        df = pandas.DataFrame(counts, columns=['word', 'count'])

        btw = df.to_csv(index = False).encode()

        fs = s3fs.S3FileSystem(key=[AWS ACCESS KEY ID]', secret=[AWS SECRET KEY ID])

        with fs.open('s3://[RESULTS BUCKET NAME]/file_test.csv', 'wb') as f:
                f.write(btw)

        sc.stop()
```

Il risultato dell'operazione di map-reduce viene trasformato in un Dataframe a 2 colonne, e poi viene sfruttata la libreria s3fs per accedere al bucket e scrivere il file csv chiamato "*file_test.cv*".
Per poter utilizzare questa parte dello script sarà quindi necessario installare la libreria s3fs.
Eseguire in locale o sulla macchina ec2 il comando:

"pip3 install s3fs"

Un'altra libreria necessaria è boto3, eseguiamo il comando nella cartella /testsite:

"pip3 install boto3"

Se nell'ambiente non è ancora presente pandas, installarlo con il comando:

"pip3 install pandas"

Dopo aver installato la libreria fare il submit al cluster dello script con la parte di scrittura su S3 non commentata, e a questo punto, a fine computazione, dovrebbe comparire il file nel bucket S3.

Se tutto è stato eseguito correttamente, sarà possibile configurare Athena.

- Nella barra di ricerca della console di gestione AWS digitare "Athena"
- Una volta aperta l'interfaccia di Athena, copiare nel text-box delle query la query:

```
CREATE EXTERNAL TABLE IF NOT EXISTS sampledb.result_table (
  `Word` string,
  `Count` int
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
WITH SERDEPROPERTIES (
  'serialization.format' = ',',
  'field.delim' = ','
) LOCATION 's3://[RESULTS BUCKET NAME]/'
TBLPROPERTIES ('has_encrypted_data'='false', "skip.header.line.count"="1");
```

Con questa query, una volta eseguita, verrà creata la tabella result_table nel database samplebd (il database di base in Athena). Il file di input per la tabella è il **file_test.csv** salvato in precedenza nel bucket specificato in RESULTS BUCKET NAME.

Una volta eseguita la query sarà stata creata la tabella da interrogare tramite la nostra applicazione Python con query SQL ad Athena.

Va detto che è possibile eseguire query su questa tabella anche direttamente dall'interfaccia di Athena, scrivendo una query SQL in uno dei box presenti. I risultati vengono mostrati in basso sotto il pannello di query.

3.3 Funzionamento applicazione di query

Una volta configurato l'ambiente Django e la tabella Athena alla quale fare le query, verrà descritto come è stata organizzata la semplice interfaccia grafica per interrogare i dati.

Per utilizzare i file presenti su GitHub nella cartella django-apps, bisogna prima di tutto copiare all'interno del proprio progetto la cartella "s3_website" all'interno della cartella "django-apps/testsite/"

In s3_website abbiamo tutti i file necessari al funzionamento della GUI che verranno descritti tra poco.

Una volta fatto questo, per permettere all'applicazione Django di eseguire il sito, sarà prima di tutto necessario modificare il file urls, dalla cartella django-apps eseguiamo:

```
"nano testsite/testsite/urls.py"
```

E sostituiamo il contenuto con:

```
from django.contrib import admin
from django.urls import path, include


urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('s3_website.urls'))
]
```

Salviamo, poi apriamo il file settings:

```
"nano testsite/testsite/settings.py"
```

Aggiungiamo in INSTALLED_APPS:

```
INSTALLED_APPS = [
[…],
's3_website.apps.S3WebsiteConfig',
'testsite'
```

```
]
```

Sempre nello stesso file sostituire TEMPLATES con:

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 's3_website/templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [

'django.template.context_processors.debug',

'django.template.context_processors.request',

'django.contrib.auth.context_processors.auth',

'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Quindi chiudiamo e salviamo il file.

Ora dovrebbe essere possibile eseguire e visualizzare il sito-web:

```
"python manage.py runserver 127.0.0.1:8000"
```

Query:

Aprendo il file "views.py" nella cartella "s3_website" sarà possibile visualizzare il sistema per effettuare le query a Athena:
I parametri delle funzioni che vedremo tra poco sono definiti in params:

```
params = {
    'region': '[YOUR REGION]',
    'database': 'sampledb',
    'bucket': '[BUCKET TO SAVE ATHENA RESULTS]',
    'path': 'Unsaved/2021/output',
    'query': 'SELECT * FROM ' + db + ' LIMIT 100'
}
```

Notiamo che ci sono alcuni parametri da definire:

[YOUR-REGION] : la regione delle macchine ec2

[BUCKET-TO-SAVE-ATHENA-RESULTS]: un nuovo, terzo, bucket, che sarà necessario creare per permettere ad Athena di salvare i risultati. Ogni volta che Athena esegue una query, infatti, viene generato un file .csv con i risultati con un nome univoco, e questo file verrà salvato nel bucket specificato. Grazie a questo salvataggio fatto da Athena, sarà possibile recuperare questo file in S3 dopo aver completato la query e visualizzare i risultati in forma tabellare sul sito web.

Le tre funzioni principali sono query, athena_query, athena_to_s3, e cleanup:

```
def query():
   session = boto3.Session()
   s3_filename = athena_to_s3(session, params)
   time.sleep(2)
   client = boto3.client('s3',
                         aws_access_key_id='[AWS ACCESS KEY]',
                         aws_secret_access_key='[AWS SECRET ACCESS KEY]', )
   # Create the S3 object
   obj = client.get_object(
      Bucket='[BUCKET TO SAVE ATHENA RESULTS]'
      Key='Unsaved/2021/output/' + str(s3_filename)
   )
   print('Unsaved/2021/output/' + str(s3_filename))
   table = pandas.read_csv(obj['Body']).to_html()
   return table


def athena_query(client, params):
   response = client.start_query_execution(
      QueryString=params["query"],
      QueryExecutionContext={
         'Database': params['database']
      },
      ResultConfiguration={
         'OutputLocation': 's3://' + params['bucket'] + '/' + params['path']
      }
   )
   return response


def athena_to_s3(session, params, max_execution=5):
   client = session.client('athena', region_name=params["region"])
   execution = athena_query(client, params)
   execution_id = execution['QueryExecutionId']
   state = 'RUNNING'

   while max_execution > 0 and state in ['RUNNING', 'QUEUED']:
      max_execution = max_execution - 1
      response = client.get_query_execution(QueryExecutionId=execution_id)

      if 'QueryExecution' in response and \
              'Status' in response['QueryExecution'] and \
              'State' in response['QueryExecution']['Status']:
         state = response['QueryExecution']['Status']['State']
         if state == 'FAILED':
            return False
         elif state == 'SUCCEEDED':
            s3_path = response['QueryExecution']['ResultConfiguration']['OutputLocation']
            filename = re.findall('.*\/(.*)', s3_path)[0]
            return filename
      time.sleep(1)

   return False



def cleanup(session, params):
   s3 = session.resource('s3', aws_access_key_id='[AWS ACCESS KEY]',
                         aws_secret_access_key='[AWS SECRET ACCESS KEY]')
   my_bucket = s3.Bucket(params['bucket'])
   for item in my_bucket.objects.filter(Prefix=params['path']):
      item.delete()
```

Sarà necessario specificare le chiavi della CLI nella funzione clenup.

Per effettuare una query viene chiamata la funzione query(). Questa funzione fa partire la sessione boto3 e chiama la funzione “athena_to_s3” passandole i parametri in params e la session. La funzione dovrà restituire il nome del file contenente i risultati della query se questa è andata a buon fine.
La funzione “athena_to_s3” inizia una sessione Athena nella regione specificata nei parametri. Chiama quindi la funzione “athena_query” alla quale passa il client e l’insieme di parametri.
La funzione “athena_query” manda la query ad Athena, sottoponendo al database specificato in ‘database’ la query specificata nel campo ‘query’ di params. Definisce anche il path nel quale trovare l’output e restituisce la risposta alla query.
Nella seconda parte della funzione “athena_to_s3” la funzione effettua 5 tentativi massimo per trovare il file csv che dovrebbe contenere i risultati della query. Se il file viene trovato, restituisce il nome.
Il nome del file verrà restituito alla funzione query(), che tramite boto3 aprirà il bucket e leggerà il file di risultati, convertendolo in Dataframe e poi in html con la funzione .to_html(), cosicché possa poi essere reindirizzato nella pagina del sito Django.
Infine, la funzione di cleanup può essere utilizzata per pulire il bucket di risultati quando il limite di spazio in S3 è vicino.

In generale, la strategia scelta per effettuare query ad Athena è sempre questa, ciò che cambia è il parametro ‘query’ in params.

Ad esempio:

```
def getMoreCommon(request):
    params['query'] = 'SELECT * FROM ' + db + ' ORDER BY count DESC LIMIT 30'
    table = query()
    return render(request, 's3_website/athena.html', {'result' : table})
```

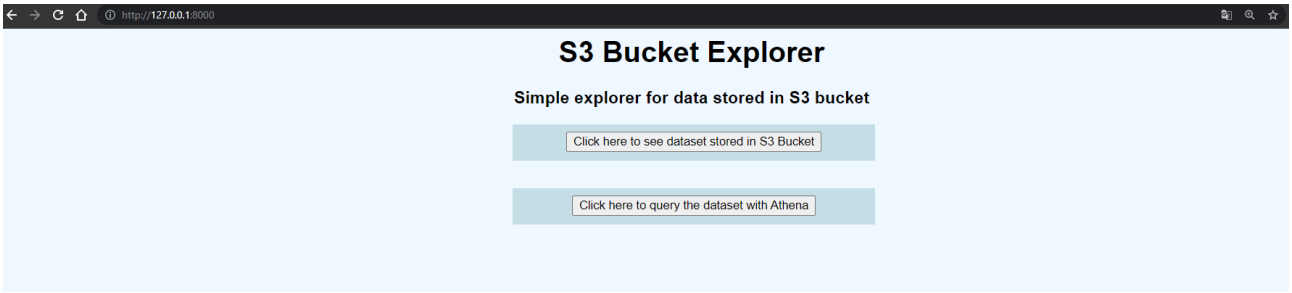
E’ la funzione che agisce sul bottone nell’interfaccia per ottenere le parole più comuni.

Nel file views.py presente nel repository è necessario riempire i campi delimitati da “[]” con la propria configurazione.

Il resto della configurazione è normale codice Django, con la definizione degli urls nel file urls.py, dei template html nella cartella templates in s3_website, e dell’intero sito in generale. Il tutto è facilmente comprensibile esplorando i file presenti nel repository GitHub.

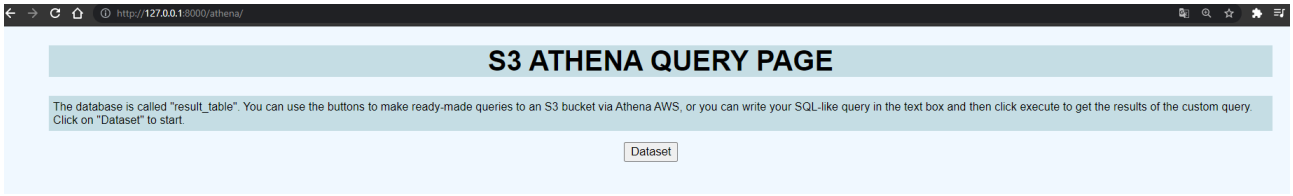
3.4 Descrizione GUI

La GUI è composta principalmente da due pagine, la prima, quella principale che si presenta all’avvio è:

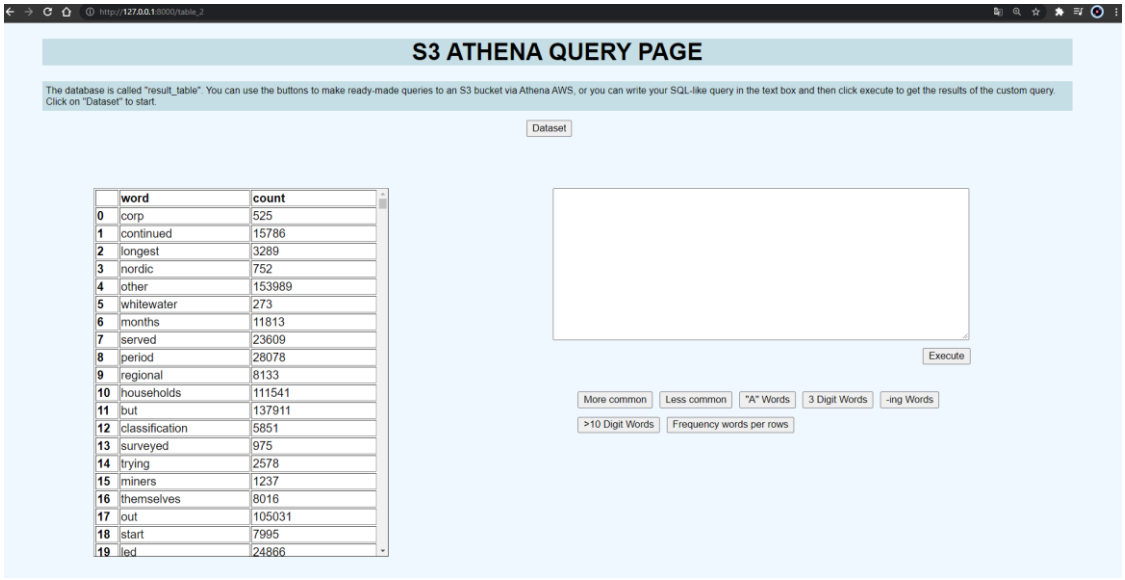


Cliccando sul primo bottone sarà possibile visualizzare in una pagina unica l'intero database di risultati.

Cliccando sul secondo bottone, invece, si aprirà la pagina delle query:

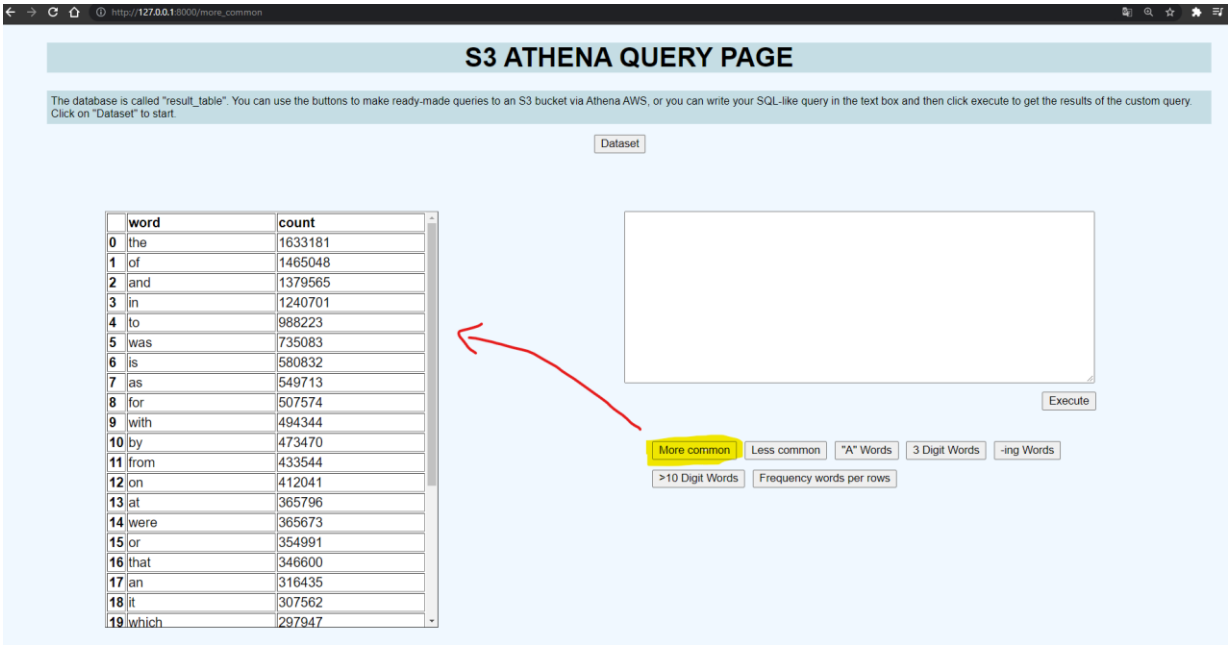


Cliccando su Dataset verrà visualizzato a destra il dataset e a sinistra i bottoni e il box per scrivere le query:



	word	count
0	corp	525
1	continued	15786
2	longest	3289
3	nordic	752
4	other	153989
5	whitewater	273
6	months	11813
7	served	23609
8	period	28078
9	regional	8133
10	households	111541
11	but	137911
12	classification	5851
13	surveyed	975
14	trying	2578
15	miners	1237
16	themselves	8016
17	out	105031
18	start	7995
19	led	24866

Se verrà cliccato uno dei bottoni sotto il textbox, la tabella a sinistra verrà sostituita con la tabella di risultati, stessa cosa se viene scritta una query SQL nel Textbox e viene successivamente cliccato execute:



	word	count
0	the	1633181
1	of	1465048
2	and	1379565
3	in	1240701
4	to	988223
5	was	735083
6	is	580832
7	as	549713
8	for	507574
9	with	494344
10	by	473470
11	from	433544
12	on	412041
13	at	365796
14	were	365673
15	or	354991
16	that	346600
17	an	316435
18	it	307562
19	which	297947