Preparación del ambiente de trabajo

Competencias

- Conocerá los principales beneficios y características de los servicios Clouds asociados al procesamiento masivo de datos.
- Aprenderá a ejecutar scripts utilizando Amazon Elastic MapReduce (EMR).
- Aprenderá las formas de interactuar con una instancia de trabajo mediante `ssh` y `scp`.

Motivación

Parte de la explosión del término "Big Data" está asociada a la irrupción del Cloud Computing como una manera más accesible de utilizar computadores con una alta capacidad de rendimiento. Mediante servicios como Amazon Web Services, Google Cloud Platform, Azure entre otros, el uso de computadoras de alta gama está al alcance de pequeñas y medianas empresas, sin la necesidad de correr con los gastos asociados al High Performance Computing On Premise.

A lo largo del curso estaremos trabajando con la plataforma Amazon Web Services donde implementaremos nuestros flujos de trabajo en Big Data. Resulta que para poder trabajar de forma óptima en la nube, es necesario tener conocimientos básicos sobre cómo interactuar con la línea de comandos. Amazon Web Services ofrece una infinidad de opciones para distintos casos de uso, pero para el contexto de Big Data, implementaremos instancias **EMR**, que definiremos posteriormente. Partamos por definir "Cloud Computing".

Definición de Cloud Computing

La Nube o Computación en la Nube (Cloud Computing en inglés) es la entrega de recursos informáticos on demand a través de Internet. Estos servicios varían desde aplicaciones convencionales hasta clusters avanzados con cientos de máquinas. Pero, ¿Por qué este concepto resulta tan relevante? Básicamente, porque le permite a las empresas dedicarse a su negocio y no preocuparse de administrar todos los sistemas y servidores necesarios para desarrollar sus actividades.

Otro beneficio de los servicios Cloud es que eliminan la inversión inicial de adquirir costosos equipos y contratar personal especializado para que los mantenga operativos. Por último, los servicios cloud son mucho más flexibles que la manera tradicional de hacer las cosas. En La Nube solo toma unos pocos minutos pasar de tener una sola máquina a tener cientos de servidores soportando la operación de un negocio. Esto permite a las empresas poder escalar rápidamente.

Así, Cloud Computing corresponde a una mejor manera de gestionar los negocios, **delegando la administración de las aplicaciones** en centros de datos de uso compartido. Los servicios Cloud presentan una serie de características atractivas tales como:

- **Disposición de recursos elásticos:** Es fácil aumentar o disminuir los recursos utilizados para satisfacer la demanda que suele ser variable.
- **Servicios cobrados por uso, no suscripción:** Paga solo por lo que usas, evitando pagar por las largas horas en que no se utilizan los equipos.
- **Autoservicio:** Existen todas las herramientas para que siga siendo posible administrar los equipos en caso de ser necesario.

En relación a los tipos de servicios que ofrece Cloud Computing que pueden ser clasificados de la siguiente manera:

- **Infraestructura como servicio (IAAS):** Aquí el proveedor de servicios ofrece una infraestructura completa junto con las tareas relacionadas con el mantenimiento.

- **Plataforma como servicio (PAAS):** En este servicio, el proveedor de la nube ofrece recursos como almacenamiento de objetos, tiempo de ejecución, cola de espera, bases de datos, etc. Sin embargo, la responsabilidad de la configuración y las tareas relacionadas con la implementación dependen del consumidor.

- **Software como Servicio (SAAS):** Este servicio es el más facilitado, que proporciona todos los ajustes necesarios y la infraestructura que ofrece IaaS para la plataforma y la infraestructura en su lugar.

Relación del Big Data y Cloud Computing

En consideración de esto, y de la lectura de la semana anterior, podemos simplificar afirmando que Big Data corresponde a lo relacionado con el procesamiento masivo de datos, mientras que Cloud Computing, a los servicios de infraestructura para soportar aplicaciones. De esta breve definición de conceptos, podemos prever que la adopción de ambos como parte de la misma solución resulta natural. La combinación de ambos produce resultados beneficiosos para las organizaciones. Por no mencionar, ambas tecnologías están en una etapa de evolución, pero su combinación ofrece una solución escalable y económica para el análisis de Big Data.

Tanto Big Data y Cloud Computing son valiosas por sí mismas. Muchas empresas apuntan a combinar las dos para obtener mayores beneficios empresariales. Ambas tecnologías apuntan a mejorar los ingresos de la empresa al tiempo que reducen el costo de inversión. Mientras que Cloud Computing administra el software, Big Data ayuda en las decisiones de negocios.

La siguiente imagen, corresponde a un modelo de cómo se relacionan Big Data y Cloud Computing:

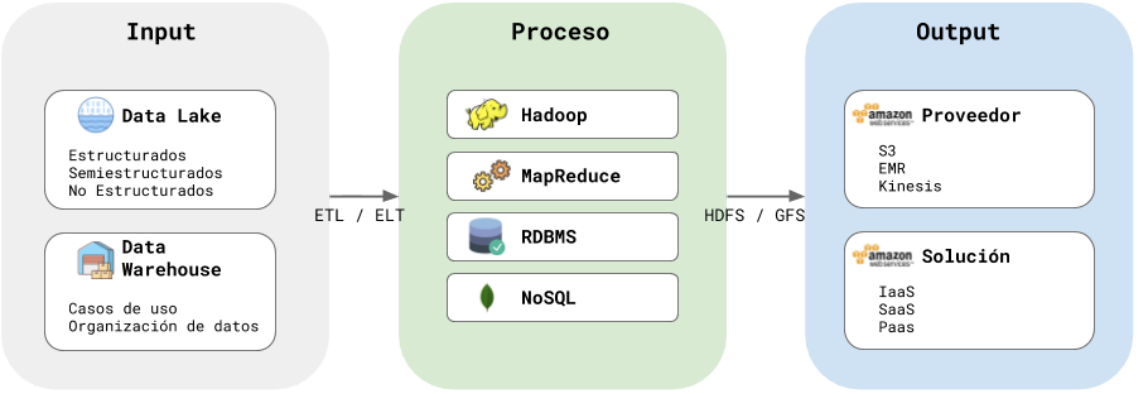

Imagen 1. Relación entre Big Data y Cloud Computing

De este modelo se puede apreciar que los datos como entrada se pueden presentar en distinta variedad utilizando distintas interfaces. Éstos son procesados en un flujo donde es necesario contar con variados motores de procesamiento de datos, cada uno con sus propias configuraciones y características. Finalmente, todo el flujo de trabajo puede ser soportado utilizando IaaS (Infrastructure as a Service), PaaS (Platform as a Service), SaaS (Software as a Service) o una solución mixta de todas estas, como se presenta a continuación:

- **IaaS en una Nube Pública:** IaaS es una solución rentable que utiliza este servicio en la nube; los servicios de Big Data permiten a las personas acceder a un almacenamiento ilimitado y poder de cómputo. Es una solución muy rentable para empresas donde el proveedor de la nube asume todos los gastos de administración del hardware subyacente.

- **PaaS en una Nube Privada:** Los proveedores de PaaS incorporan las tecnologías de Big Data en el servicio ofrecido. Por lo tanto, eliminan la necesidad de lidiar con las complejidades de administrar elementos de software y hardware individuales, lo cual es una preocupación real al tratar con terabytes de datos.

- **SaaS en una Nube Híbrida:** Analizar datos de redes sociales es hoy en día un parámetro esencial para las empresas en el análisis de sus negocios. En este contexto, los proveedores de SaaS proporcionan una excelente plataforma para realizar el análisis.

Amazon Web Services

A lo largo del curso implementaremos Amazon Web Services como nuestro proveedor de servicios en la nube. En específico, haremos uso de los servicios **AWS Elastic Map Reduce (EMR)** para generar instancias de trabajo **AWS EC2** preconfiguradas, y **AWS S3** como nuestro sistema de alojamiento de documentos e información. Partamos por definir los elementos que componen cada uno de estos servicios.

Amazon EC2

Para lograr entender cómo funciona nuestro motor EMR, es necesaria una breve definición de EC2. EC2 representa un componente principal de los servicios de AWS, dado que son las máquinas virtuales que ocupamos para distintas aplicaciones. Toda máquina virtual EC2 se conoce como una instancia de trabajo, donde se contendrán todos los programas y elementos necesarios para que una aplicación funcione. Estas máquinas virtuales son configurables en cuanto a la cantidad de recursos asignados (en Disco Duro, Procesadores, Memoria RAM) y software a instalar. Dado que éstas son agnósticas a la implementación que le dará el usuario, Amazon Web Services provee del servicio Amazon Machine Image, una serie de imágenes preconfiguradas para una serie de tareas.

Amazon ElasticMapReduce (EMR)

Posteriormente veremos sobre lo complejo que es la instalación del suite Hadoop Ecosystem en una máquina. Con el objetivo de reducir el tiempo en configuración e instalación de paquetes asociados a Big Data, AWS generó ElasticMapReduce. Esta es una plataforma preconfigurada que simplifica el uso de frameworks analíticos de Big Data como Apache Hadoop y Apache Spark. Por defecto, EMR utiliza el ecosistema Hadoop para distribuir y procesar los datos en clusters escalables. Amazon EMR es compatible con 19 proyectos de código abierto diferentes, incluidos Hadoop, Spark, HBase y Presto, con computadoras portátiles EMR administradas para la ingeniería de datos, el desarrollo de la ciencia de datos y la colaboración. Cada proyecto se actualiza en EMR dentro de los 30 días posteriores a la publicación de la versión, lo que garantiza que tenga lo último y lo mejor de la comunidad, sin esfuerzo.

Amazon S3

Hasta el momento tenemos definido cómo funciona nuestra instancia de trabajo específica con EMR. Por lo general, no se guardan archivos en la misma máquina EC2. Para ello implementaremos S3 para alojar archivos. Por lo general, podemos entenderlas como una especie de Dropbox o Google Drive, guardando archivos de manera rápida sin la necesidad de preocuparse por detalles técnicos. Dado que se consideran una capa más abstracta, las instancias de S3 no se conocen como Volúmenes (a diferencia de nuestros discos físicos en el computador local), pero se conocen como buckets.

Configuración de la cuenta

A lo largo del curso trabajaremos con AWS Educate, servicio por el cual tendremos una serie de créditos gratis a nuestro uso. Una vez que ingresen al portal, diríjanse a "My Classrooms", donde encontrarán su curso. Hagan click en el botón "Go To Classroom". Si todo sale bien, accederán a un portal donde les saldrá este cuadro de información:

Imagen 2. Estado de cuenta.

Siempre deben tener en consideración este cuadro, dado que les entregará el acceso a la consola AWS (en el botón azul AWS Console) y los detalles de la cuenta (en el botón azul Account Details). Este último botón lo utilizaremos cuando utilicemos la línea de comandos de AWS, `aws cli`, para administrar servicios.

Creando nuestro bucket con Amazon S3

Ahora exploremos cómo funciona el servicio de almacenamiento de AWS mediante S3. Si hacemos click en el botón `AWS Console`, nos redireccionará a la consola de administración general de AWS (AWS Management Console). Esta funciona como nuestra página maestra en la administración completa de nuestros servicios asociados. La vista principal tendrá la siguiente composición:

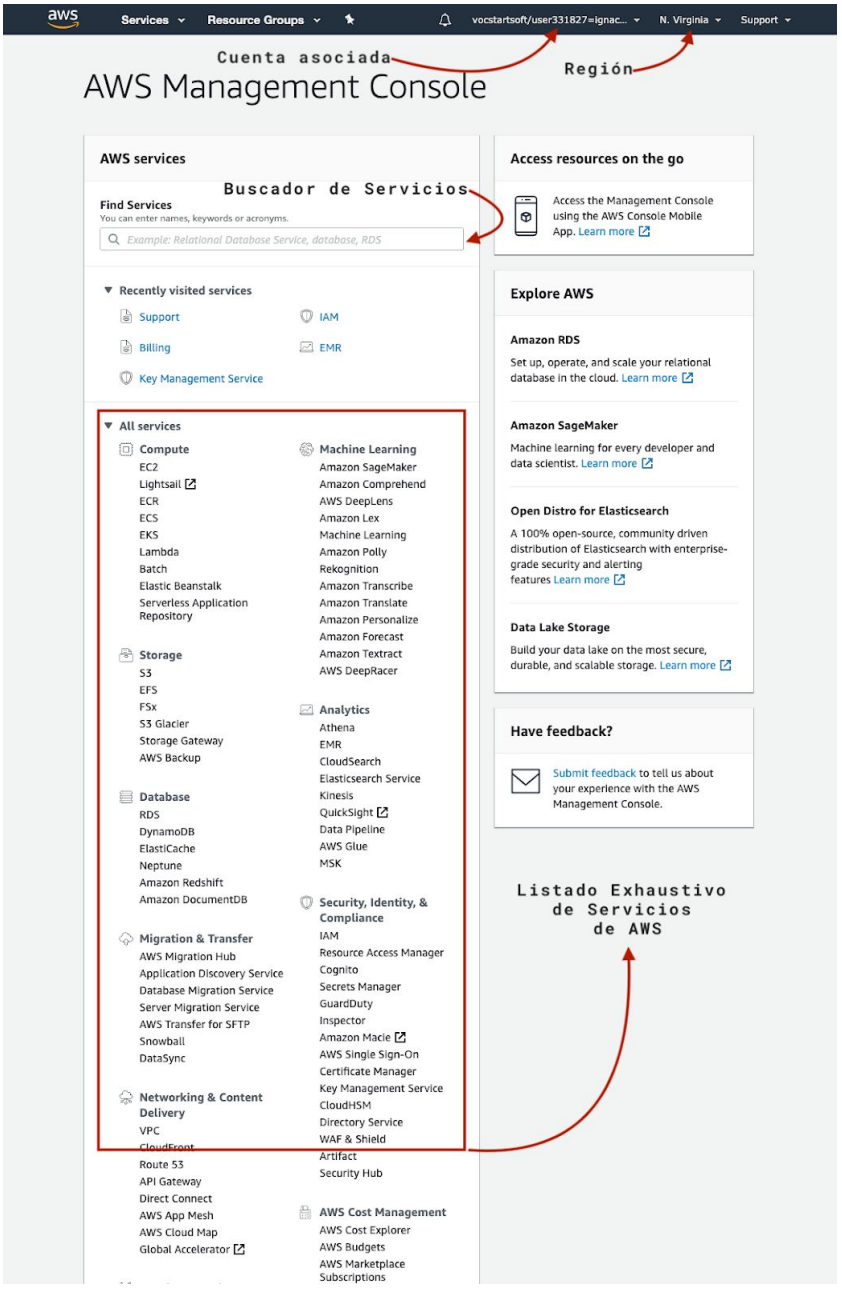

Imagen 3. Vista principal AWS Management Console.

La vista nos entregará la región específica donde se encuentra nuestro servidor, así como nuestro nombre de usuario y una lista exhaustiva de servicios provistos en AWS. En el buscador "Find Services" podemos buscar los servicios de una manera más rápida.

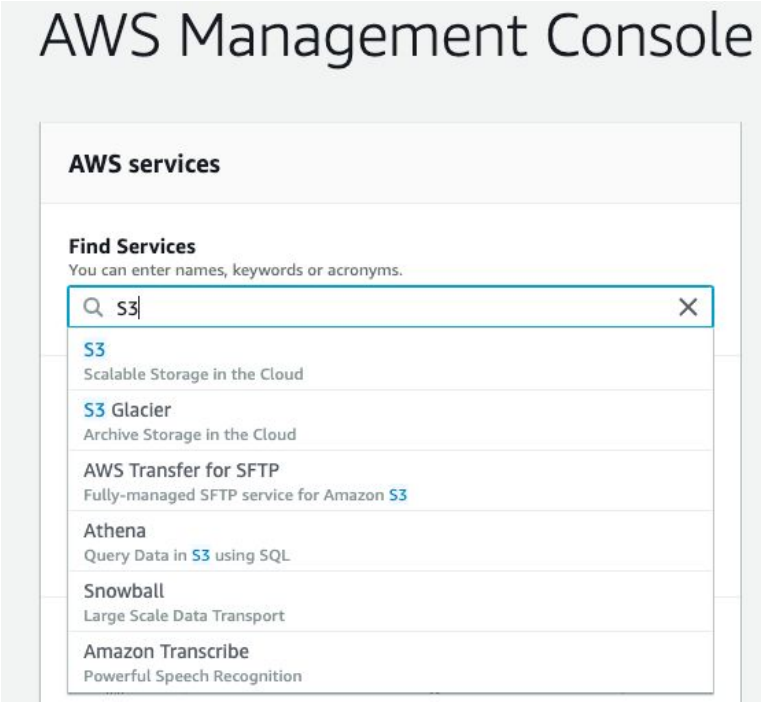

Imagen 4. Buscador “Find Services”.

Si escribimos S3 y lo seleccionamos, nos redireccionará a una nueva página que entrega la siguiente información. Haremos click en `+Create Bucket` para tener nuestro objeto de almacenamiento.

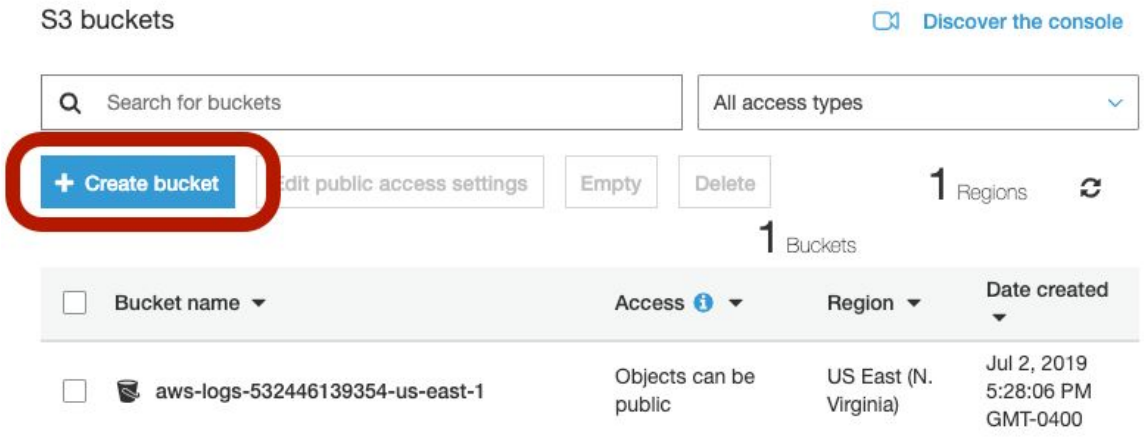

Imagen 5. Objeto de almacenamiento.

Al hacer click, nos abrirá una caja que nos solicita la siguiente información:

1. En el paso 1 (*Name and region*) deben generar un nombre único a nivel global. Por ahora, ignoren la casilla regiones.

2. Por el momento, no hay elementos a modificar en el paso 2 (*configure options*).

3. El tercer paso (*Set permissions*) nos permite definir si los datos alojados podrán ser públicos o estarán asociados a mi cuenta de manera específica. Por ahora no hagamos nada.

4. En el último paso (*Review*) confirmamos que nuestra información sea correcta.

Si todo sale bien, hemos creado nuestro primer bucket y éste se encuentra listado en el controlador de S3. Si lo seleccionamos, podemos subir archivos varios. Un bucket S3 puede funcionar perfectamente como una arquitectura de Data Lake (manteniendo una serie de precauciones asociadas). Ahora vamos a generar y subir un script de Python donde podamos simular datos:

```
#script_demo.py

"""
simula una cantidad n de observaciones que siguen una especificación:

age - Un número entero aleatorio entre 18 y 90
income - Un número entero aleatorio entre 10000 y 1000000.
employment_status - Un string entre Employed con probabilidad de
ocurrencia .7 y Unemployed con probabilidad de ocurrencia .3.
debt_status - Un string entre Debt con probabilidad de ocurrencia .2 y
No Debt con una probabilidad de ocurrencia de .8
churn_pr - Probabilidad predicha de churn para el usuario siguiendo una
distribución BetaBinomial(alpha=600, beta=300).

uso:
python3.6 script_demo.py 1

donde 1 hace referencia a un número identificador.

"""
```

```
import random
import numpy as np
import csv
import sys


catch_number = sys.argv[1]

def create_random_row():
   age = random.randint(18, 90)
   income = random.randrange(10000, 1000000, step=1000)
   employment_status = np.random.choice(['Unemployed', 'Employed'], 1,
[.3, .7])[0]
   debt_status = np.random.choice(['Debt', 'No Debt'], 1, [.2, .8])[0]
   churn_pr = np.random.beta(600, 300)

   return [age, income, employment_status,debt_status, churn_status]

with open(f"simulate_churn_{catch_number}.csv", 'w') as csvfile:
   file = csv.writer(csvfile, delimiter=',', quotechar='|',
quoting=csv.QUOTE_MINIMAL)
   for _ in range(10000):
       file.writerow(create_random_row())

print("Script Listo!")
```

Pasamos por los pasos Select Files → Set Permissions → Set properties → Review, y nuestro archivo debería estar en la nube. Al actualizar la información, nuestro archivo debería estar dentro de nuestro bucket. Si hacemos click en éste, tendremos información sobre cómo acceder con el tag **Object URL** y el botón `Copy path`. El primero nos permitirá acceder a este mediante el navegador, y la segunda forma nos permitirá acceder mediante la línea de comandos. La primera opción no funcionará dado que por defecto nuestro archivo es privado.

AWSCLI - La línea de comando de AWS

Ahora accederemos al contenido en nuestro bucket mediante la línea de comando. Para ello, vamos a abrir nuestra consola e instalarlo mediante anaconda con la línea `conda install -c conda-forge awscli`. Una vez que ya esté instalado, el primer paso es configurar nuestra información de la cuenta de AWS con la línea de comando. Tengan a mano los detalles de su cuenta, que pueden acceder en cuadro **Your classroom account details** en el portal principal, donde deberían ver una información similar a:

Imagen 6. Ejemplo Credenciales.

Y haciendo click podrán acceder a:

```
AWS Access
   Session started at: 2019-07-02T13:54:31-0700
   Session to end  at: 2019-07-02T16:54:31-0700
   Remaining session time: 2h58m58s

   Term: 59 days 04:34:57

AWS CLI:
   Copy and paste the following into ~/.aws/credentials
   [default]
   aws_access_key_id=ASI[redacted]HGR
   aws_secret_access_key=RPBn[redacted]Zq3
   aws_session_token=FQoGZXIvYXdzEI7//////////wEaDDof7bYIZNyAHsNYbiKDA0wTUhtiDB7Uo4UwixxfRaK2wQvff2Bx
   rIhPEiHshGxZQ/Y0dKg90TYitFFGGnEy+RiRTaCUhRSsAueXb3MlWlOgIRUAstaV+KFn8civRz8Ge8tMLuQ0ZJHMMx57/C5jca
   KQFpRG2afx5WIFXmJGaiKY33NjwlLXxuo44pW3orXLwo8nk0/Ju/5+U1UQBz+lK3pkZso2ketHNL9kShOzh4Oi8j0w4bRkiEND
   jJnw/QqUGDN9ajg9SNSRcglKhWYD7Ln71S1rWB2XcbmWcLAT6bKiXIQcxTk0Fo66zb+vI8tmDQvwVxrVC3K/a8lt7ifYWmdGoQ
   MomsenfPFLGDcdIUdwe1EWH9V55w0X1WMKYs4ySgZzlYAWZfkpTapklRGGSgX0ks5tVbzeoQMyymkgCFKW0qP+DQ1B+HjpA6Q6
   fBEcXXSahZ2HO2999wjS5J+LM5y1yrWn+8jraBbLOHPLC9LVCWnv2M2zThCnAzQiPWM++ZUuQB1Y5r65RXWPHR3Y0fMcHiiIi
   e/oBQ==
```

`aws_access_key_id`: Id para ingresar a la API.

`aws_secret_access_key`: "Contraseña" asociada a Id.

Imagen 7. Show API.

Para ello, dentro del terminal escribiremos:

```
aws configure

AWS Access Key ID [None]: <ingresamos nuestro aws_access_key_id> <enter>
AWS Secret Access Key [None]: <ingresamos nuestro
aws_secrect_access_key> <enter>
Default region name [None]: <enter>
Default output format [None]: <enter>
```

Posteriormente, vamos al archivo `credentials` que se encuentra en la carpeta .aws dentro de nuestro home y pegamos la última llave `aws_session_token` al final del archivo. Ahora estamos en capacidad de listar todos los archivos dentro de nuestro bucket ocupando la sintaxis:

```
aws s3 ls s3://iszprimerbucket/
# RETORNO => 2019-07-04 17:44:48       645 script_demo.py
```

¿Qué fue lo que hicimos acá? La primera palabra `aws` hace referencia que vamos a estar operando desde el comando AWS. La segunda palabra `s3` especifica que vamos a realizar alguna acción con los bucket `s3`. La tercera palabra es un comando de Linux que nos permitirá listar todos los elementos dentro de un directorio. Finalmente, especificamos el directorio con la siguiente ruta `s3://iszprimerbucket/`.

Digresión: Una serie de comandos de Linux y sus análogos en AWS

De aquí en adelante, haremos uso prevalente de la línea de comando para interactuar con los servicios de AWS y con el ecosistema de Hadoop. Algunas de las acciones más comunes están asociadas con el dónde nos situamos dentro de un computador, así como copiar, mover y eliminar archivos. En la siguiente tabla se presentan sus implementaciones más comunes. Cabe destacar que `<local>` hace referencia a dónde se posicionan actualmente en el computador local.

Acción	En Linux	En AWS S3
Ver dónde estoy ubicado	`pwd`	
Cambiar de directorio	`cd <ruta>`	
Listar elementos en el directorio.	`ls -lah`	`aws s3 ls s3://<ruta>`
Copiar archivos entre directorios	`cp archivo.txt ruta_destino`.	`aws s3 cp <local> s3://<ruta>`
Copiar carpetas entre directorios	`cp -r carpeta ruta_destino`	`aws s3 cp <local> s3://<ruta> --recursive`
Mover archivos entre directorios	`mv archivo.txt ruta_destino`	`aws s3 mv <local> s3://<ruta>`
Eliminar archivos	`rm archivo.txt`	`aws s3 mv <local> s3://<ruta> --recursive`
Eliminar carpetas	`rm -rf carpeta`	`aws s3 rm <local> s3://<ruta>/archivo`
Ver contenidos en archivo	`cat archivo`.	`aws s3 rm <local> s3://<ruta>/archivo --recursive`.

Tabla 1. Comandos de interacción.

Si quisiéramos bajar de nuevo el archivo `script_demo.py`, podríamos ejecutar:

```
# esta línea permite copiar desde un bucket a nuestro punto actual
aws s3 cp s3://iszprimerbucket/script_demo.py .

#RETORNO => download: s3://iszprimerbucket/script_demo.py to
./script_demo.py
```

Donde el `.` hace referencia a dónde nos encontramos. Si todo sale bien, el retorno indicará que se descargó el archivo.

Subiendo múltiples archivos a nuestro bucket S3 desde la consola.

Para ejercitar el uso de la línea de comando entre nuestro computador y nuestro bucket, nuestro objetivo va a ser generar 10 copias del `csv` con nuestro script de Python mediante la línea de comandos. A grandes rasgos, el procedimiento implica:

1. Crear una nueva carpeta `datos_simulados`, mover nuestro script `script_demo.py` a esa carpeta.

```
# creamos una carpeta con el comando mkdir
mkdir datos_simulados

# movemos el script desde nuestra posición original a la nueva carpeta
con mv
mv script_demo.py datos_simulados
```

2. Dentro de la carpeta `datos_simulados`, implementar un loop que llame 10 veces el script y guarde los datos.

```
# nos cambiamos a nuestra carpeta
cd datos_simulados

# implementamos el loop donde i es el iterable
for i in {1..10}; do; python script_demo.py $i; done
```

3. Crear una carpeta `export_data` y mover todos los `csv` generados.

```
# creamos otra carpeta
mkdir export_data

# mediante el asterisco consideramos a todos los archivos con extension
csv y los movemos a la carpeta creada
mv *.csv export_data
```

4. Exportar la carpeta a una nueva carpeta en el bucket creado. El output del comando nos debería avisar `upload`.

```
# con la opción --recursive logramos considerar la carpeta.
#
aws s3 cp export_data s3://iszprimerbucket/datos_simulados --recursive

upload: export_data/simulate_churn_4.csv to
s3://iszprimerbucket/datos_simulados/simulate_churn_4.csv
upload: export_data/simulate_churn_3.csv to
s3://iszprimerbucket/datos_simulados/simulate_churn_3.csv
upload: export_data/simulate_churn_5.csv to
s3://iszprimerbucket/datos_simulados/simulate_churn_5.csv
upload: export_data/simulate_churn_7.csv to
s3://iszprimerbucket/datos_simulados/simulate_churn_7.csv
upload: export_data/simulate_churn_6.csv to
s3://iszprimerbucket/datos_simulados/simulate_churn_6.csv
upload: export_data/simulate_churn_2.csv to
s3://iszprimerbucket/datos_simulados/simulate_churn_2.csv
upload: export_data/simulate_churn_8.csv to
s3://iszprimerbucket/datos_simulados/simulate_churn_8.csv
upload: export_data/simulate_churn_10.csv to
s3://iszprimerbucket/datos_simulados/simulate_churn_10.csv
upload: export_data/simulate_churn_1.csv to
s3://iszprimerbucket/datos_simulados/simulate_churn_1.csv
upload: export_data/simulate_churn_9.csv to
s3://iszprimerbucket/datos_simulados/simulate_churn_9.csv
```

Finalmente, podemos listar todos los elementos dentro de nuestro bucket con `aws s3 ls`.

```
aws s3 ls s3://iszprimerbucket/datos_simulados/

2019-07-04 18:59:11     348754 simulate_churn_1.csv
2019-07-04 18:59:11     349001 simulate_churn_10.csv
2019-07-04 18:59:11     348949 simulate_churn_2.csv
2019-07-04 18:59:11     349246 simulate_churn_3.csv
2019-07-04 18:59:14     348582 simulate_churn_4.csv
2019-07-04 18:59:12     349383 simulate_churn_5.csv
2019-07-04 18:59:11     348647 simulate_churn_6.csv
2019-07-04 18:59:11     349025 simulate_churn_7.csv
2019-07-04 18:59:11     349016 simulate_churn_8.csv
2019-07-04 18:59:11     349130 simulate_churn_9.csv
```

Creando nuestro primer cluster con Amazon ElasticMapReduce

Si S3 es un contenedor con un limitado rango de acciones, EMR (y EC2) es un computador virtual mediante el cual podremos implementar nuestro flujo de trabajo. Partamos por visitar todos los pasos necesarios para levantar nuestro primer computador virtual.

Preliminar: Crear un permiso key pair con extensión **pem**

Dado que estaremos interactuando con un computador que se encuentra en la nube, es necesario generar una llave que permita establecer una conexión segura entre nuestro computador local y el servidor remoto. De esta manera, todas las instrucciones generadas desde nuestro computador, se podrán implementar sin mayor problemas en la instancia AWS EMR.

Para ello, necesitamos crear un **key pair** con Amazon EC2. En el AWS Management Console, nos dirigimos a la consola de administración EC2.

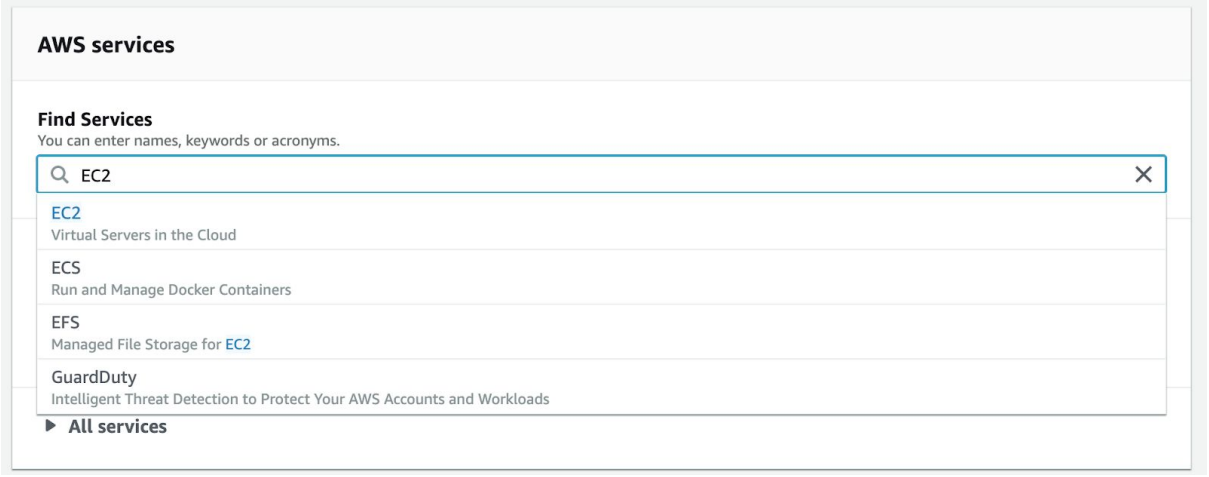

Imagen 8. EC2,

Dentro de la consola, accedemos a la opción **Key Pairs** que se encuentra en Network & Security.

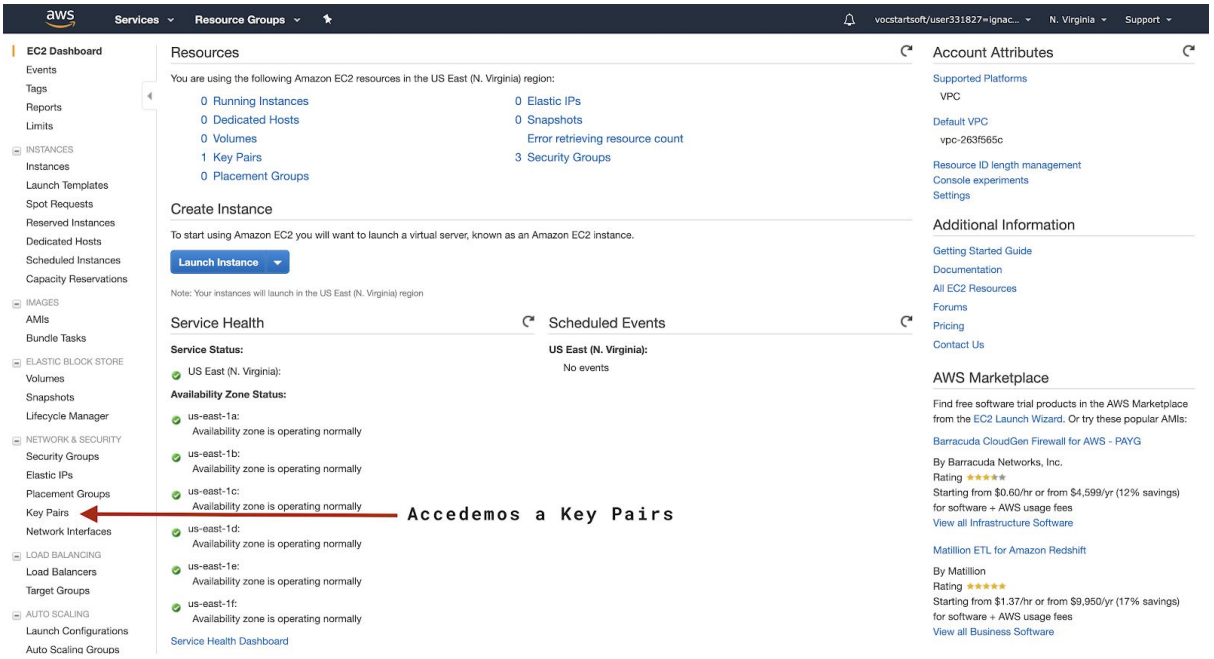

Imagen 8. Key Pairs.

Esta opción nos listará todas las llaves existentes. Hacemos click en el botón `Create Key Pair`.

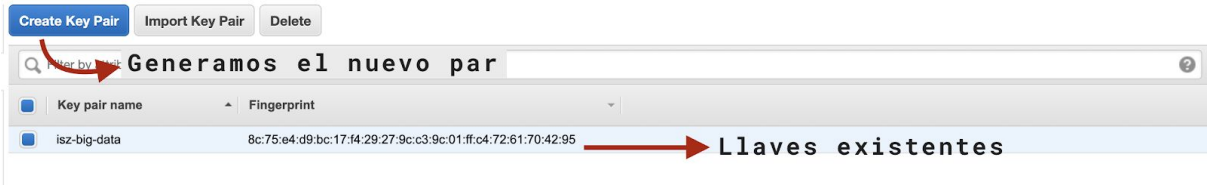

Imagen 10. Creando Key Pair.

Debemos generar un nombre que sea lo suficientemente indicativo sobre cuál es la tarea a realizar. Si bien podemos implementar la misma llave para absolutamente todos los servicios, diferenciarlas por uso no hace daño.

Imagen 11. Generar nombre.

Al confirmar la creación de la llave, se generará la descarga automática. Un punto importante a considerar es que debemos recordar de dónde dejamos la llave. Una buena práctica es dejarlas en nuestro home (~ en Linux y Mac). Si revisamos el contenido de la llave, observaremos una serie de caracteres que no nos hacen sentido, pero sí al servidor.

```
isz :: ~ » more bigdata-desafiolatam.pem
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAg8Zhz00EiYI2J3YnC46f4ED7apl77J5Y/p5SsoVkw2nJ+/mceIRPJ7miYIVs
FmKxMVl69UWOn6vf+lnPE7SZRBPOdkCxtVu4alYoxmZS2XAQCMnMoPLisN+VOHWiwOl8Na1YDP45
LI7flr7pF4yGYkm3E3d+BFyoCXWNdZpbQTue8tm5luNBOrI6t1OD02Mia2sesbiPiXwZw4zaqRPz
MESWjs7bwDLAi7/spIYODRFSwJQUHzVqEiPjtNcTk6mZh0MbQIAjg6PiZGdqZa5J81UBxWYT9neT
4Ov4YpN/q6ReRZ93uLRrbznNtac9Tu68g0MnIqBuaHxk8ps3Qoy/uwIDAQABAoIBABozCSNUJsDI
OMgjv6OWmcbWZKZPGcpzuTH2fG6uAtivYc9NU4TKmYpGaDhUjqbsvFoZdk7po2+vlnqJ/JfoYkic
v4HXXv213LaXpse7dGFy3F1C8gJWCCgua9ar3M4KHf32FvmOWgO37FBmaKWb9OP50AJ40X9AhNBM
pPFq3RXc3fp+mnCOyaD/X57I+/GmFADb4yONsqbkvEI4E4TtAVPqhobnABtW9PP3OrQxl1JsVXcb
kNgOvCfibNf3CSbYM5lIwsZlD0pOLuCA212VAH7LqoIl4drtFDFUtZlyOGyYqNQuE1aDeFBC7URb
xxfjV1FJyIlVpTRQHy35n3b2OAECgYEA/Im8tHApmovH+cNrMEcqJS9c5mNphVJpZsGMGIeux+ND
ya88/KwKXB1fF1517V6ISDoV3TapJFHmLosvtw45aZja5PaXwJE7LJw56/C6uoR2Xf/sAFas9y7z
oaltT9YrznQzTK1KHtSHZN+hDVTsBoQu1iFcz702aIchI+YAwgECgYEAhZTWCXNnIVLogz0PmyD9
MM8iiPKyuX5nUNUzh/66Rlsw7Tk1ZJfFheQ3mcE0U69eBDXDVV5cA+vinCoCgFwTDUi8nUFiXG/E
XA92TucNxI/9xjJQOoJhmMH7+Y3j3HmyxHSXrQ6A6pI5bsiQq4X3kSpmIT2nxPs/8dkNSB8tCbsC
gYAtJWJ6z2JgooSJfgn3g4FhTGpLNZJzpxv4prko2XjBeztvHsjEUu03MnkBcH2chb3PS7ME7PSn
7Yw83d0V2JEi85SOQhc28b+mZfdk2pBHL88JVusV1Gjvd5FOYT+NejgynEtfdcs+X7gkrGXaQEhB
kffdh/FTofvMC/w8sq2IAQKBgCVTnq+LYNEOAszRXjAB+t4Ns1/DOa/do28R9FFXG6QXTUesib/v
LRGp4OOE+mM8CLByeX+8W0f5PRxYHNTTZbt7jMEAiUS9MohrBasso0s+1THx0ztWPYFbC1cDkB2G
oC6HSe7gs6NYctYDE3HqgHA/AcTeHqhLcOXaufxiQcqLAoGBAMtGlSf1cjxG/ZgxOG9hS5LyT//m
qXt9Ganbc+FnSQJvNJWlxK8f6Byu85hItqLN6kMzg7n5RbevN1T5jCR08IJzt7P6b53PxYJG7UOp
lyKVYYKZeak7L60uqJeoaG086V/vDkZAFWNvlVSj9vZwwlNnftrLD+DQhas3+1XY03eu
-----END RSA PRIVATE KEY-----
bigdata-desafiolatam.pem (END)
```

Imagen 12. Contenido de la llave.

Ya tenemos nuestro archivo key pair generado, el cual implementaremos más adelante en AWS EMR.

1. Ir a AWS Management Console y buscar EMR en "Find Services".

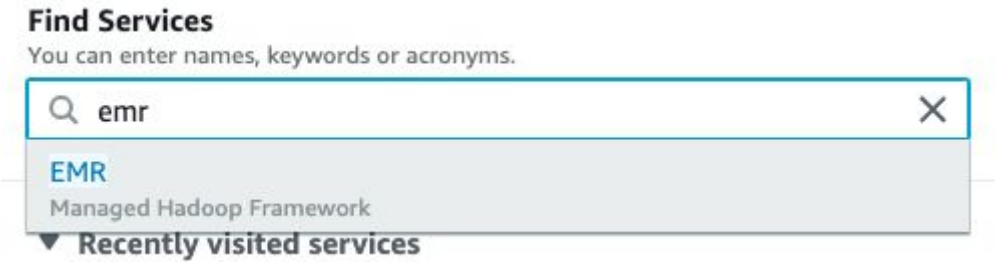

Imagen 13. Buscar EMR.

2. Entrar a AWS EMR

Dentro de la consola de Amazon EMR, observaremos un historial de clusters que contendrán información sobre el status (Ejecutado, Terminado, En Espera) e indicadores sobre su mantención y tiempo de ejecución. También encontraremos una serie de botones que permiten acciones a nivel de cluster, **Create Cluster** (Crear una nueva instancia de trabajo desde cero), **View Details** (Ver detalles de la configuración del cluster), **Clone** (Clonar una instancia de trabajo previa con la misma configuración) y **Terminate** (terminar la instancia de trabajo).

Imagen 14. Consola Amazon EMR.

3. Hacemos click en "Create cluster" y accedemos a la configuración general

Al hacer click en **Create cluster**, AWS EMR nos llevará a una nueva vista con una serie de pasos. Por motivos expositivos, vamos a dividirla en cuatro fases:

3.1 Configuración general

En la configuración general, vamos a ingresar un nombre **indicativo** sobre la misión de nuestra instancia de trabajo en la casilla **Cluster name**. Bajo esta casilla encontraremos la opción Logging que creará un bucket S3 donde se guardará toda la información de los procesos. Esto puede ser útil cuando tengamos algún problema y deseemos ver qué pasó. Finalmente, la opción **Launch Mode** permite configurar si el cluster va a funcionar de manera independiente o formará parte de un flujo de trabajo mayor. Por el momento dejémoslo en `Cluster`.

En caso de no tener identificados todos los pasos de ejecución de los distintos scripts, es recomendable utilizar la opción Launch mode: `Cluster`. Bajo la opción `Step execution` puedes dejar tu script en un bucket de S3, ejecutarlo e incluso escoger los parámetros de ejecución. Esta opción es recomendada cuando has corrido un script muchas veces y quieres ahorrar el ingresar a la máquina y ejecutar tú mismo el script. Recuerda que cada minuto que el cluster está en funcionamiento posee un costo asociado.

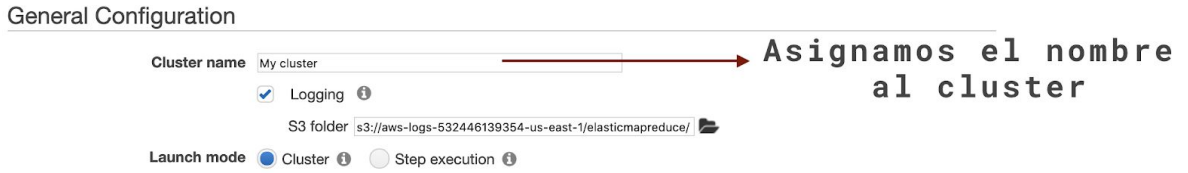

Imagen 15. Configuración general.

3.2 Configuración de software

Posterior a la definición del nombre y tipo de cluster que tendremos, procederemos a incluir una serie de alternativas de software. Es en este punto donde radica la importancia de Amazon Elastic Map Reduce: **Viene con una serie de configuraciones predeterminadas las cuales nos ahorran el proceso de instalación y configuración del ecosistema de trabajo con Hadoop.**

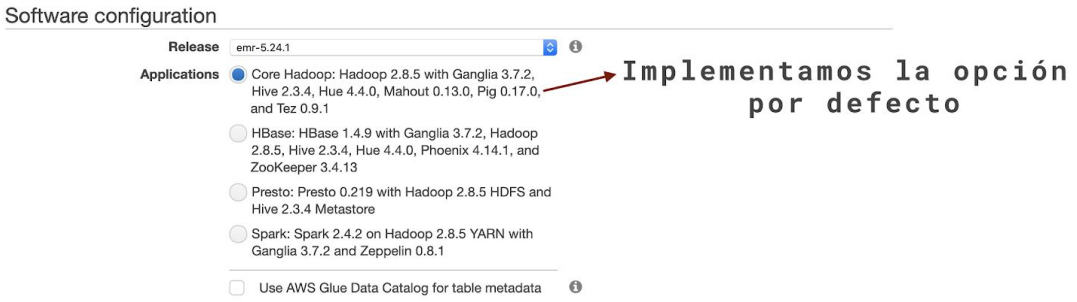

Imagen 16. Configuración de software.

Por el momento quedémonos con la configuración por defecto `Core Hadoop`.

3.3 Configuración de hardware

En la opción Hardware Configuration debes escoger las tipo de cluster que deseas arrendar. Si es uno con mayores capacidades, será más caro. El precio asociado a cada tipo de cluster lo encuentras en el siguiente link. Es también en esta opción donde configuras la cantidad de nodos que tendrá el cluster.

Imagen 17. Configuración de hardware.

3.4 Configuración de seguridad

Ahora debemos vincular nuestro cluster con la llave key pair que generamos anteriormente. Para ello, seleccionamos del menú **EC2** key pair la llave con el nombre creado anteriormente.

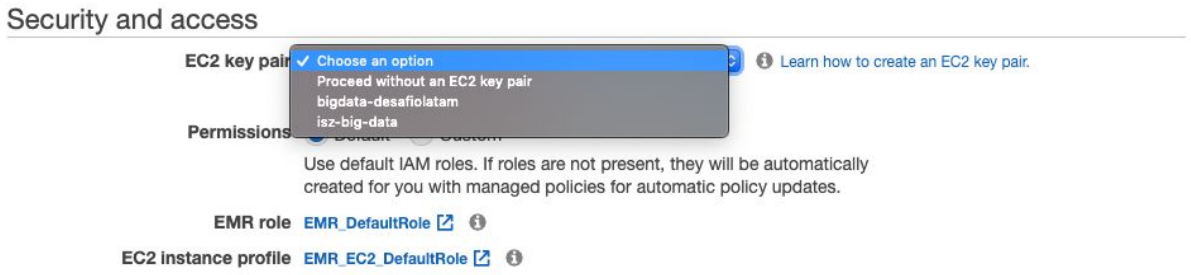

Imagen 18. Configuración de seguridad.

Una vez que tengamos la configuración lista, hacemos clic en el botón `Create cluster`.

Trabajando con nuestro cluster creado en Amazon EMR

A continuación, presentaremos los principales elementos a considerar en la administración de nuestro cluster recién creado.

Imagen 19. Elementos en la administración de un cluster.

Una vez que confirmemos la creación del cluster, AWS nos enviará a una vista de administración sobre la instancia recién creada. En la imagen de arriba, se han encapsulado los principales puntos a considerar cuando estemos en esta vista.

- **Status:** El sector indicado con morado representa cuál es el estado actual de nuestra instancia de trabajo. Cuando iniciemos nuestro cluster, este tendrá la palabra Starting en verde, lo que significa que se está realizando el proceso de asignación de nuestra instancia en los servidores de AWS, así como la instalación de nuestro ambiente de trabajo preconfigurado que definimos en el punto 3.2. Una vez que el cluster esté listo, la palabra será Waiting, con lo que estaremos habilitados de interactuar con la instancia.

- **Acciones:** El sector indicado con verde representa a las posibles acciones a realizar con nuestro cluster creado. Lo podremos clonar, terminar y generar un script para su creación desde AWS CLI.

- **Conexiones:** El sector indicado con azul representa las formas mediante podremos interactuar con nuestra instancia de trabajo. El campo **Connections** hace referencia a cómo podemos interactuar mediante interfaz gráfica en nuestros navegadores web, asunto que visitaremos posteriormente. El campo **Master public DNS** nos entrega una dirección por la cual nosotros podemos ingresar a nuestra instancia de trabajo mediante la línea de comando. Si hacemos click en el link `SSH`, saldrá la siguiente caja:

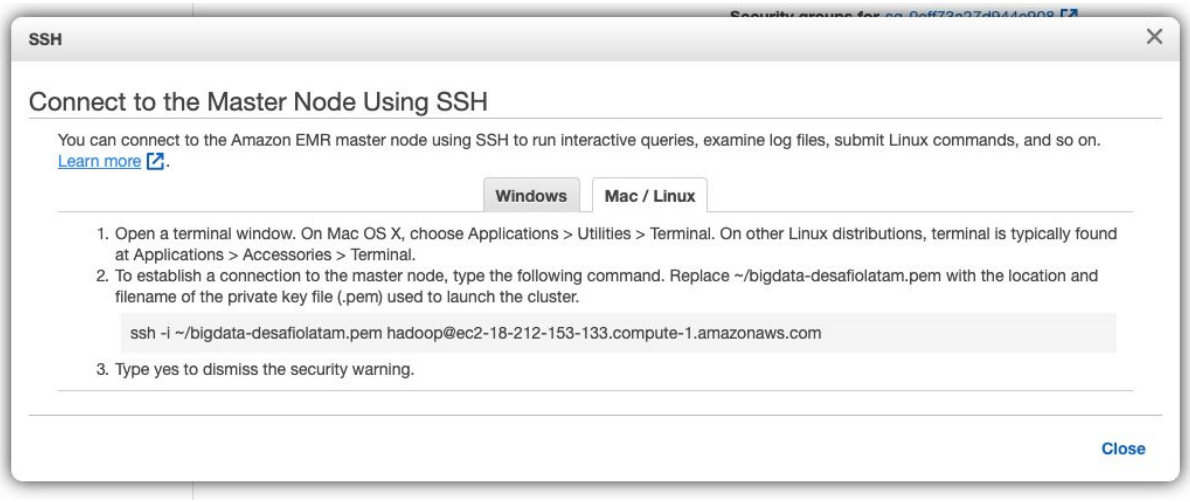

Imagen 20. Conección usando SSH.

La línea entregada se puede entender de la siguiente manera:

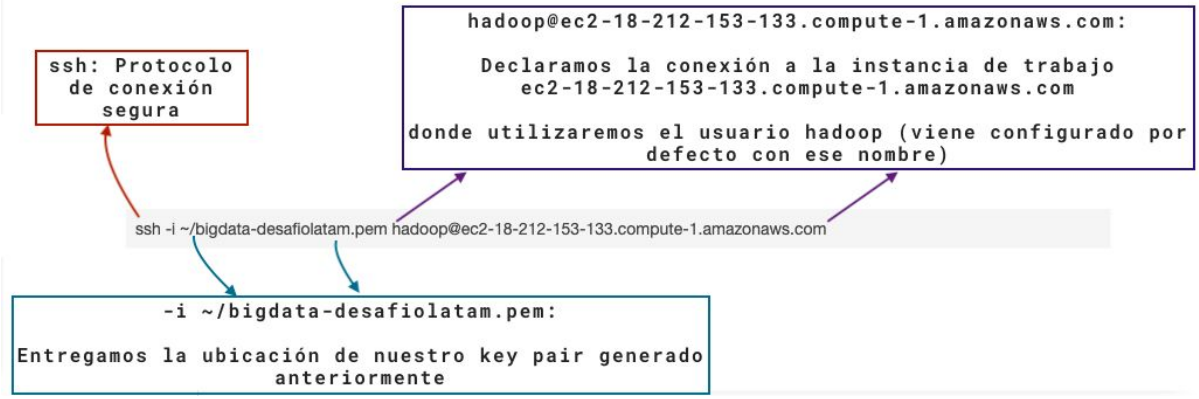

Imagen 21. Partes de la entrada.

- **Estado de los nodos:** El sector indicado con amarillo reportará cuál es la salud y estado de nuestros nodos **Master** y **Core**. El campo **Availability zone** señalará en dónde se aloja nuestra instancia.

- **Configuración de seguridad:** El sector indicado con rojo aloja las configuraciones de seguridad. Ahí aparece nuestro Key pair y otras opciones.

Una vez que el status de nuestro cluster cambie a Waiting, estamos en capacidad de poder interactuar con este. Vayamos al terminal:

```
isz :: ~ » #Veamos dónde estoy actualmente
isz :: ~ » pwd
/Users/veterok
isz :: ~ » #Estoy en mi carpeta HOME. Ahora revisemos si tengo mis permisos. Para ello voy a ocupar el comando ls, seguido de un pipe |
isz :: ~ » #Donde seleccionaré todos los archivos que contengan las letras pem
isz :: ~ » ls -lah | grep pem
-rw-------@   1 veterok  staff   1.7K Jul  3 11:53 big-data-geeneral.pem
-rw-------@   1 veterok  staff   1.7K Jul  2 17:22 bigdata-desafiolatam.pem
-rw-------@   1 veterok  staff   1.7K Jul  2 11:50 isz-big-data.pem
isz :: ~ » #Perfecto. Teniendo en cuenta mi permiso, puedo ocuparlo.
```

Imagen 22. Terminal.

Si ejecutamos la línea `ssh -i ~/bigdata-desafiolatam.pem hadoop@ec-54-91-244-151.compute-1.amazonaws.com` la primera vez, probablemente nos encontraremos con el siguiente error que indica la falta de permisos a nivel de usuario para implementar el archivo `pem`.

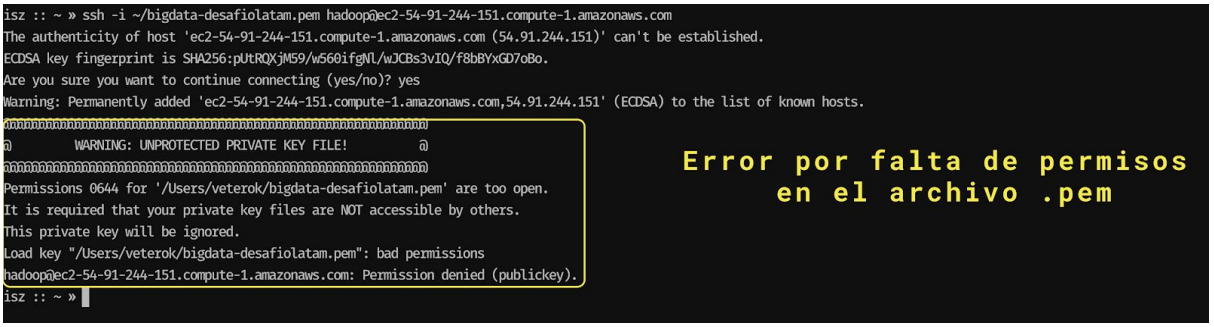

Imagen 23. Error por permisos.

Para otorgar permisos al archivo, es necesario modificarlo con chmod `og-rwx bigdata-desafiolatam.pem`. Si volvemos a ejecutar la línea `ssh -i ~/bigdata-desafiolatam.pem hadoop@ec-54-91-244-151.compute-1.amazonaws.com` en nuestro terminal tendremos la siguiente imagen, la cual nos confirma la conexión:

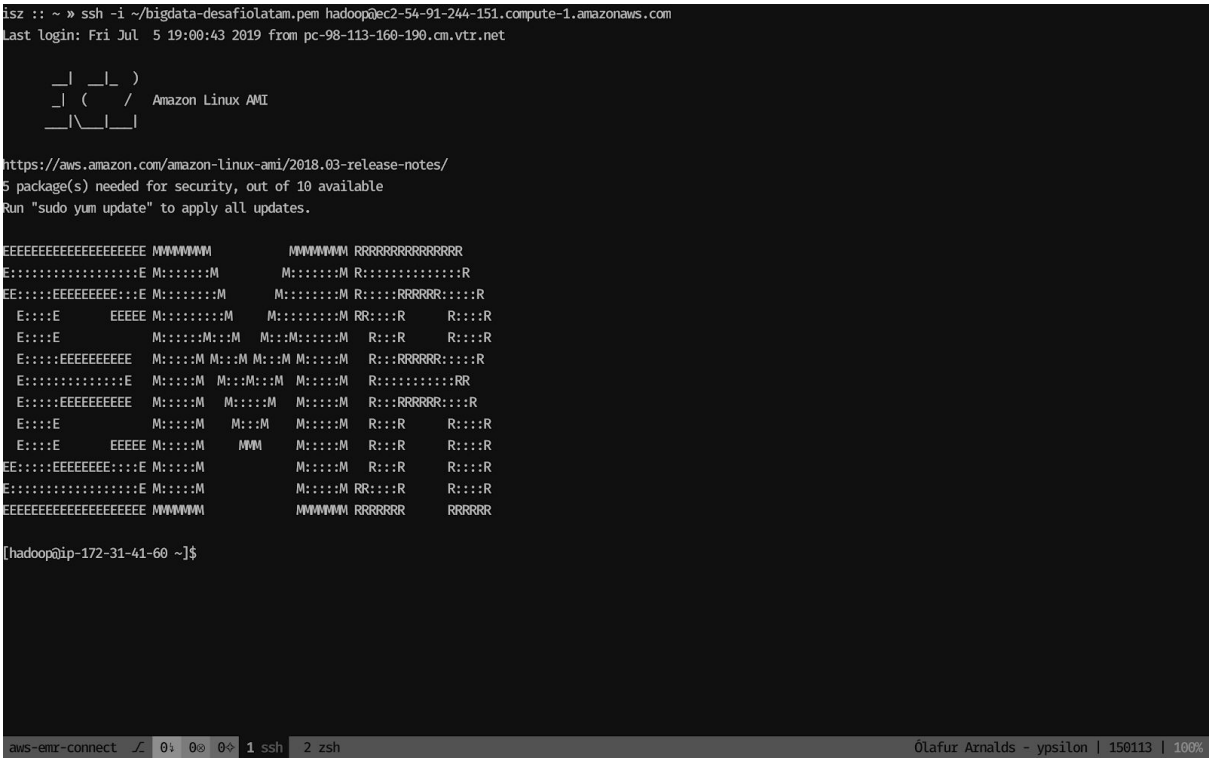

Imagen 24. EMR.

La instancia iniciada será completamente en Linux, por lo cual podremos realizar instalaciones y ejecutar comandos. Por lo general nuestro trabajo depende fuertemente de las librerías de Python instaladas. Lamentablemente, por defecto no vienen las librerías más comunes como `pandas`, `numpy` y `scikit-learn`, pero podemos instalarlas con el siguiente comando:

```
sudo pip-3.6 install ipython pandas numpy scikit-learn
```

Es importante destacar dos elementos. En primer lugar, nuestro comando comienza con `sudo`, que significa "Ejecuta lo siguiente como un Super Usuario". Si no lo implementamos, la instancia de AWS EMR no podrá ejecutar de forma exitosa el comando. El segundo elemento a considerar es `pip3.6`, que es el gestor de librerías en Python 3.6, la versión específica que hemos ocupado a lo largo del bootcamp.

Una vez instaladas las librerías necesarias, estamos en pie de poder seguir trabajando con nuestra instancia. Tomemos el siguiente ejemplo de cómo implementar un árbol de clasificación simple dentro de iPython. Ignorando deliberadamente el hecho que **nunca evaluamos nuestro modelo en el mismo conjunto de datos**, su implementación no dista de lo que aprendimos en Machine Learning:

```
[hadoop@ip-172-31-41-60 ~]$ ipython
Python 3.6.8 (default, May 24 2019, 18:27:52)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.6.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from sklearn.datasets import load_iris

In [2]: from sklearn.tree import DecisionTreeClassifier

In [3]: data = load_iris(); X = data.data; y = data.target

In [4]: tree_clf = DecisionTreeClassifier().fit(X, y)

In [5]: from sklearn.metrics import classification_report

In [6]: print(classification_report(y, tree_clf.predict(X)))
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00        50
           2       1.00      1.00      1.00        50

    accuracy                           1.00       150
   macro avg       1.00      1.00      1.00       150
weighted avg       1.00      1.00      1.00       150
```

Imagen 25. Ejemplo de árbol de clasificación.

Uniendo las partes locales y remotas en nuestro trabajo

Por lo general cuando trabajamos con instancias EC2, nuestro flujo implica desarrollar scripts de preprocesamiento/análisis de manera local, los cuales subiremos a nuestra instancia de trabajo y posteriormente ejecutaremos de manera remota en un conjunto de datos. Finalmente, cuando el procesamiento esté listo, podemos rescatarlo a nuestro local y eliminar la instancia de trabajo. Así, podemos definir el flujo de trabajo en los siguientes pasos:

1. Desarrollar un script y subirlo a EC2.
2. Consumir datos desde algún bucket o subirlos a nuestra fuente local.
3. Ejecutar nuestro script en EC2.
4. Guardar resultados

1. Desarrollar un script y subirlo a EC2

Partamos por desarrollar nuestro script. Para ello vamos a trabajar con los datos existentes en nuestro bucket creado anteriormente. En mi caso, los datos se encuentran en la ruta `s3://iszprimerbucket/datos_simulados`, y estos fueron creados con `script_demo.py` de manera local y posteriormente subidos a la ruta detallada.

Supongamos que nuestro objetivo es desarrollar un modelo predictivo para ver si un individuo caerá en deuda o no. Para ello vamos a implementar un modelo Logístico manteniendo las demás columnas de nuestros csv creados como atributos. El script tendrá más o menos la siguiente forma:

```
# run_models.py

# realizamos los imports necesarios
from sklearn.metrics import classification_report
from sklearn.linear_model import LogisticRegression
import pandas as pd
import glob, os

print("Training model on first dataset")
```

```
# ingestamos los datos, declaramos que no tenemos etiquetas de columna
en el csv.
# para efectos prácticos, vamos a entrenar con el primer batch de datos.
df = pd.read_csv('simulate_churn_1.csv', header=None)
# separamos nuestro vector objetivo
y = df.iloc[:, 3]
# de nuestros atributos
X = df.drop(columns=[3])

# entrenamos el modelo (ojo, sin dividir datos)
clf = LogisticRegression().fit(X, y)

# para cada csv existente
for i in glob.glob(os.getcwd() + '/*.csv'):
    # imprimimos el log.
    print("Validating on :", i)
    # separamos nuestro vector objetivo
    y = df.iloc[:, 3]
    # de nuestros atributos
    X = df.drop(columns=[3])
    # generamos un reporte con el clasificador entrenado.
    print(classification_report(y, clf.predict(X)), '\n')
```

Ya con nuestro script guardado, podremos subirlo con el comando `scp` en la línea de comando. scp significa Secure Copy, y funciona de una manera similar a cómo lo hace `cp`: Debemos especificar un archivo/directorio a copiar y especificar dónde vamos a alojarlo. La parte Secure viene del hecho que vamos a incorporar nuestro archivo `pem` para validar la transacción entre nuestro local y la instancia de trabajo remoto. La implementación se ve a continuación:

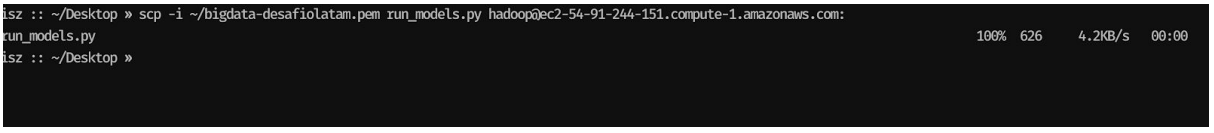

Imagen 26. Implementación de `cp`.

A continuación se presenta una tabla resumen con los principales casos de uso del comando `scp`.

Acción	Comando
`scp archivo usuario@servidor:/ruta/de/destino`	Copiar archivo desde local al servidor.
`scp usuario@servidor:/ruta/de/origen /local/ruta/de/destino`	Copiar archivo desde el servidor a local.
`scp -i mykeypair.pem archivo usuario@servidor:/ruta/de/destino`	Copiar archivo desde local a servidor, proveyendo una archivo de permiso para validar la transacción.
`scp -i -r mykeypair.pem directorio usuario@servidor:/ruta/de/destino`	Copiar un directorio desde local a servidor, proveyendo un archivo de permiso y declarando que la copia será recursiva con `-r` (todos los elementos dentro del directorio se considerarán).

Tabla 2. Usos de comando `scp`.

2. Consumir datos desde algún bucket o subirlos a nuestra fuente de manera local

Nuestro script solo tendrá sentido en la medida que tenga datos para ser ejecutados. En este ejercicio vamos a realizar la forma más rudimentaria de incorporar los datos en nuestra instancia: Mediante la copia de archivos.

Cabe destacar el siguiente punto: **Por lo general se prioriza no copiar archivos de datos en nuestra instancia EC2, dado que existen formas de poder consumir los datos directamente desde el bucket**. Por efectos expositivos vamos a realizar la copia de archivos de manera rudimentaria, pero es un punto a tener en consideración.

Para copiar los archivos, podemos utilizar el comando `aws s3 cp s3://<ruta> . --recursive` **desde nuestra instancia de trabajo**. Por lo general el comando `aws` ya vendrá con las credenciales configuradas, por lo que no es necesario incorporarlas tal como lo hicimos en nuestro computador local.

```
[hadoop@ip-172-31-41-60 ~]$ aws s3 cp s3://iszprimerbucket/datos_simulados/ . --recursive
download: s3://iszprimerbucket/datos_simulados/simulate_churn_1.csv to ./simulate_churn_1.csv
download: s3://iszprimerbucket/datos_simulados/simulate_churn_3.csv to ./simulate_churn_3.csv
download: s3://iszprimerbucket/datos_simulados/simulate_churn_10.csv to ./simulate_churn_10.csv
download: s3://iszprimerbucket/datos_simulados/simulate_churn_2.csv to ./simulate_churn_2.csv
download: s3://iszprimerbucket/datos_simulados/simulate_churn_4.csv to ./simulate_churn_4.csv
download: s3://iszprimerbucket/datos_simulados/simulate_churn_6.csv to ./simulate_churn_6.csv
download: s3://iszprimerbucket/datos_simulados/simulate_churn_7.csv to ./simulate_churn_7.csv
download: s3://iszprimerbucket/datos_simulados/simulate_churn_9.csv to ./simulate_churn_9.csv
download: s3://iszprimerbucket/datos_simulados/simulate_churn_5.csv to ./simulate_churn_5.csv
download: s3://iszprimerbucket/datos_simulados/simulate_churn_8.csv to ./simulate_churn_8.csv
[hadoop@ip-172-31-41-60 ~]$
```

Imagen 27. Archivos copiados.

Si todo sale bien, aws nos informará del upload de cada archivo.

3. Ejecutar nuestro script en EC2

El script asume que tanto los `csv` como el script se encuentran en la misma ruta para ser ejecutados. Otro punto relevante es que debemos especificar la versión de Python que implementaremos. Así, si ejecutamos `python3.6 run_model.py` tendremos nuestro modelo entrenado y validado en partes. Los Warnings que aparecen abajo representan avisos de depreciación y de métricas mal definidas, cosas que en un trabajo detallado deberíamos evaluar. Con la opción `> output_performance.txt` nos permite atajar el output de los modelos en un archivo que podremos bajar.

```
[hadoop@ip-172-31-41-60 ~]$ clear                                                                                    [33/1788]
[hadoop@ip-172-31-41-60 ~]$ python3.6 run_models.py > output_performance.txt
/usr/local/lib64/python3.6/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to si$
ence this warning.
  FutureWarning)
/usr/local/lib64/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in $
abels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib64/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in $
abels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib64/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in $
abels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib64/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in $
abels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib64/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in l
abels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib64/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in l
abels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib64/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in l
abels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib64/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in l
abels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib64/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in l
abels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/usr/local/lib64/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in l
abels with no predicted samples.
 aws-emr-connect  0  0  0  1 [tmux]  2 nvim                              Full Of Hell - Armory of Obsidian Glass | 173715 | 100%
```

Imagen 28. Warnings.

4. Exportar resultados

Finalmente, podemos extraer el archivo `output_performance.txt` mediante `scp`, e imprimir los datos con `head`.

```
isz :: ~/Desktop » scp -i ~/bigdata-desafiolatam.pem hadoop@ec2-54-91-244-151.compute-1.amazonaws.com:/home/hadoop/output_performance.txt .
output_performance.txt                                                                    100% 3680     3.3KB/s   00:01
isz :: ~/Desktop » head output_performance.txt
Train model on first dataset
/home/hadoop/simulate_churn_1.csv

              precision    recall  f1-score   support

           0       0.50      1.00      0.67      4991
           1       0.00      0.00      0.00      5009

    accuracy                           0.50     10000
   macro avg       0.25      0.50      0.33     10000
isz :: ~/Desktop »
```

Imagen 29. Datos impresos.

Terminar servicio AWS EMR

Ya terminamos nuestro primer trabajo con Elastic MapReduce, y podemos ver el comportamiento del modelo en una serie de archivos `csv` que teníamos alojados en el bucket `s3`. Para finalizar, debemos terminar de ejecutar nuestra instancia. Para ello debemos ir a la consola de AWS EMR, seleccionar el cluster que está en ejecución, que generalmente estará con un círculo verde y realizar lo siguiente:

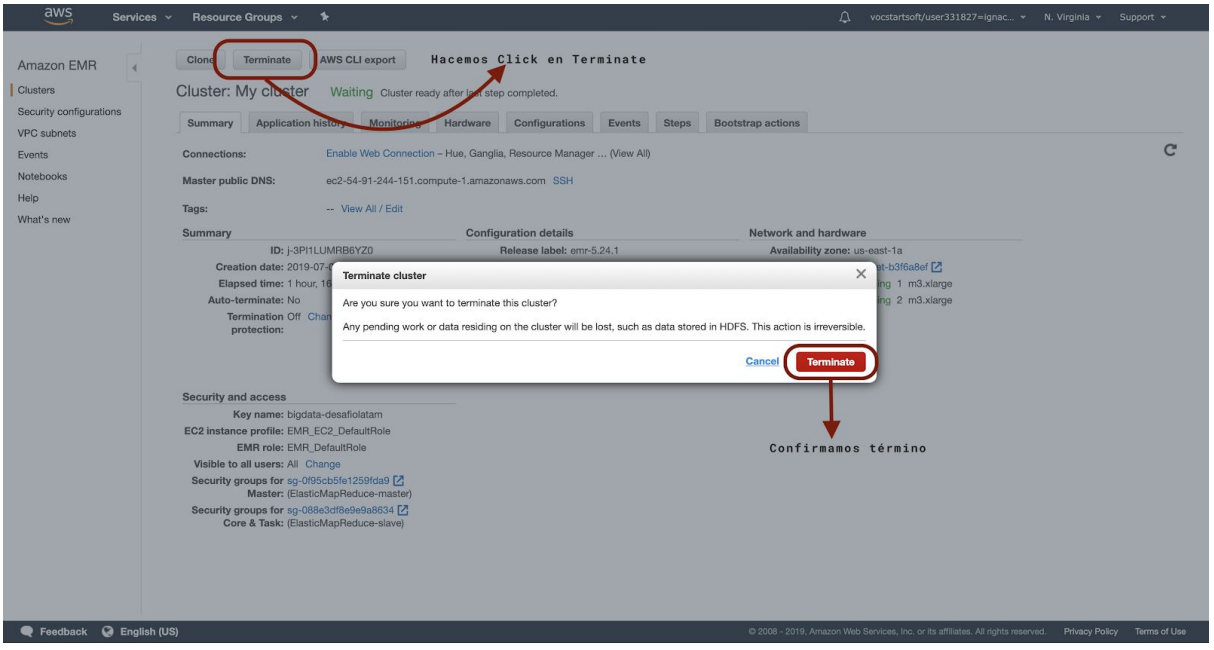

Imagen 30. Servicio AWS EMR finalizado.

Anexo: Algunas recomendaciones

Todos los comandos que vimos dependen mucho de la acción que deseemos ejecutar. A continuación se presenta un diagrama que aclara el proceso en base a la interacción que deseemos establecer con los servicios creados.

Por lo general, nuestra interacción siempre partirá por habilitar algún servicio AWS. Una vez iniciando la instancia o el bucket de alojamiento. Teniendo conocimiento de la ruta de acceso a los servicios, podemos comenzar con la interacción entre nuestro computador local y nuestra instancia de trabajo remota.

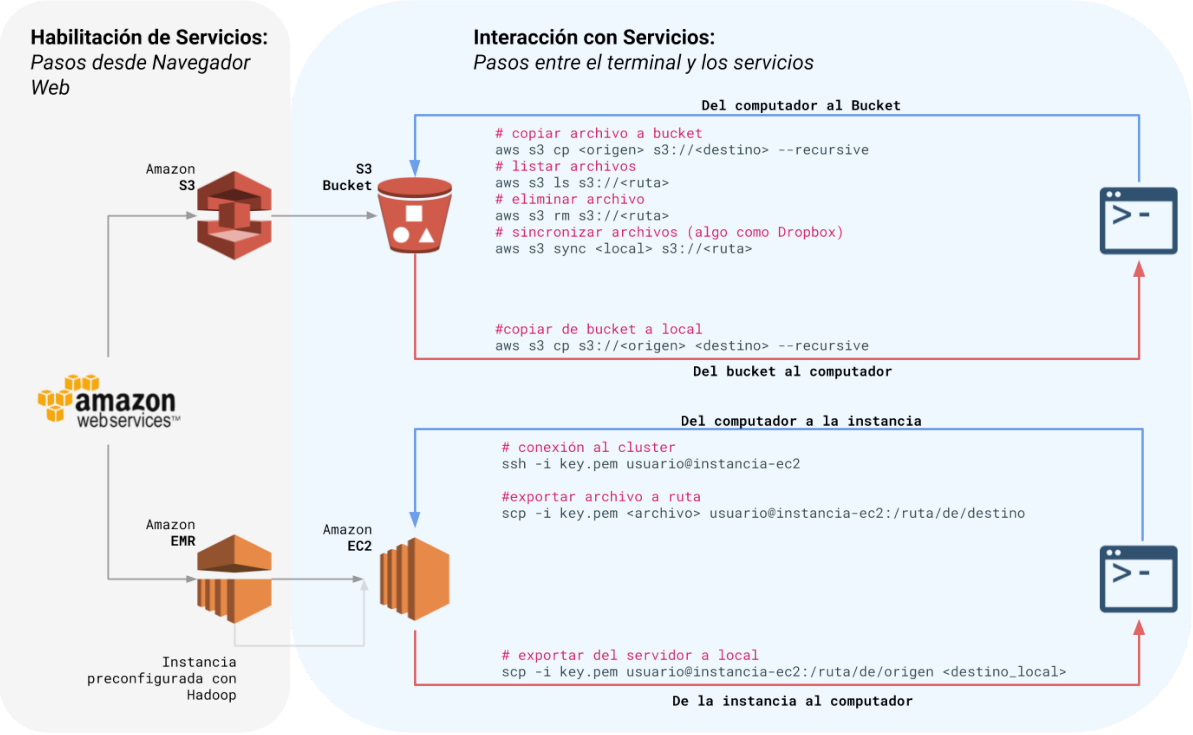

Imagen 31. Habilitación e Interacción con Servicios.

Anexo: Un checklist de sanidad mental

Trabajar con servicios cloud es complejo, suelen tener muchos pasos y a veces nos podemos confundir. A continuación se presenta una serie de puntos a considerar cuando trabajemos con estos servicios

1. [ ] Dejen algún indicador (un dibujo, sticker, luz, etc...) cerca de su computador **para recordarse que están trabajando con servicios cloud**.

2. [ ] (Sólo para cuando estén trabajando con la cuenta AWS Educate): Busquen los datos de AWS CLI y cópialos sobreescribiendo el archivo `~/.aws/credentials` de su computador local. Esto se debe a que cada vez que iniciamos una sesión en AWS Educate, nuestras credenciales son reinstauradas.

3. [ ] Fíjense en qué llave PEM utilizarán, y dónde estará ubicada. Por lo general es bueno dejar las llaves en el Home de su computador, para tener un acceso más rápido.

4. [ ] Si tienen un bucket s3 con datos, anoten su dirección `s3://dirección-a-utilizar/nombre-carpeta`.

5. [ ] Configuren una instancia de trabajo EMR con los requerimientos necesarios. Pueden seleccionar alguna configuración por defecto o especificar los programas necesarios.

6. [ ] Inicien su servidor y mantengan dos ventanas en el terminal: Una para interactuar mediante `ssh` y `scp` y otra para habilitar los servicios web asociados como Hue, JupyterHub, etc...

7. [ ] Una vez que terminen de procesar los datos, descarguen la información procesada a su computador de manera local.

8. [ ] **¿Se acuerdan del punto 1?** Ese indicador les recordará siempre que antes de terminar de trabajar con su servicio, **DEBEN TERMINARLO**.