Hadoop y Map Reduce en AWS.

José Incera, Abril 2019

Introducción

En esta práctica nos familiarizaremos con una de las arquitecturas de cómputo distribuido más populares en la actualidad: El sistema Hadoop. Utilizaremos el paradigma de programación MapReduce.

- En la primera parte, desplegaremos una instancia de Hadoop en un solo nodo en Amazon EC2.
- En la segunda, ejecutaremos algunos programas bajo el paradigma MapRedice en se nodo.
- En la tercera parte, extenderemos nuestro ambiente a un cluster de cuatro nodos.
- En la cuarta y última parte, correremos algunos programas MapReduce en nuestro cluster.

Objetivos

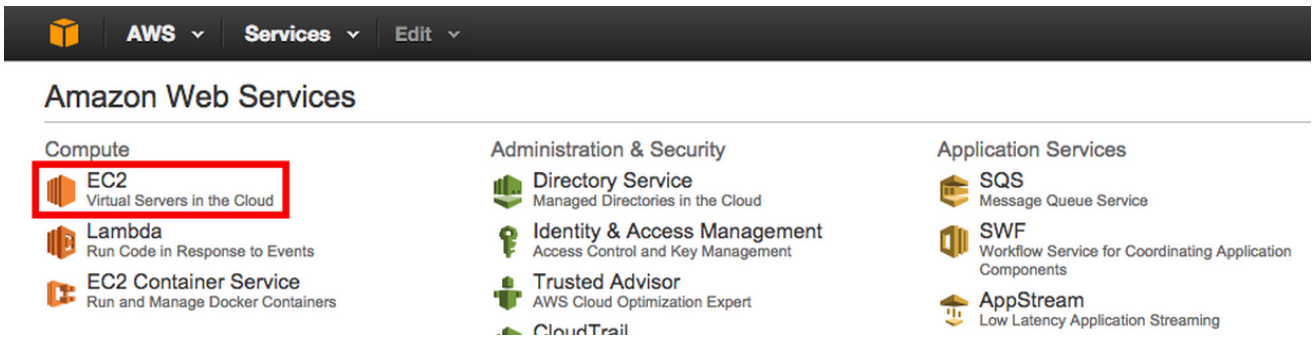
- Reafirmar conceptos básicos de la arquitectura Hadoop y el modelo de programación MapReduce.
- Desplegar un cluster de Hadoop en la nube Amazon EC2

1. Instalación de un nodo Hadoop en AWS EC2

1.1 Crear una instancia EC2 en AWS

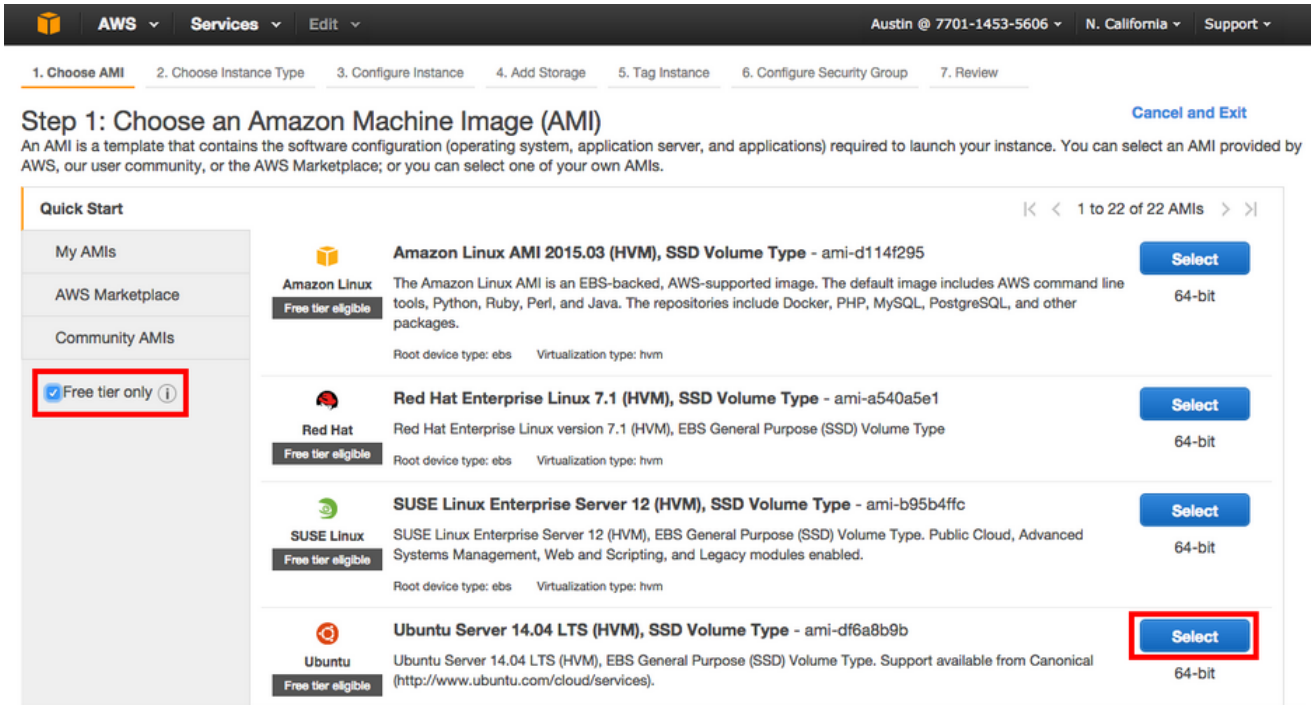
1.- Ingrese a su cuenta AWS. Si aún no tiene una, es un buen momento para crearla en aws.amazon.com Los servicios gratuitos son más que suficientes para nuestro proyecto.

2.-De clic en EC2 (*Elastic Cloud Compute*) y lance una máquina virtual



3.- Seleccione la región (puede ser cualquiera)

4.- Para la imagen de la máquina virtual, seleccione *Ubuntu Server 14.04 LTS (HVM)* y de clic en *Free tier only.*



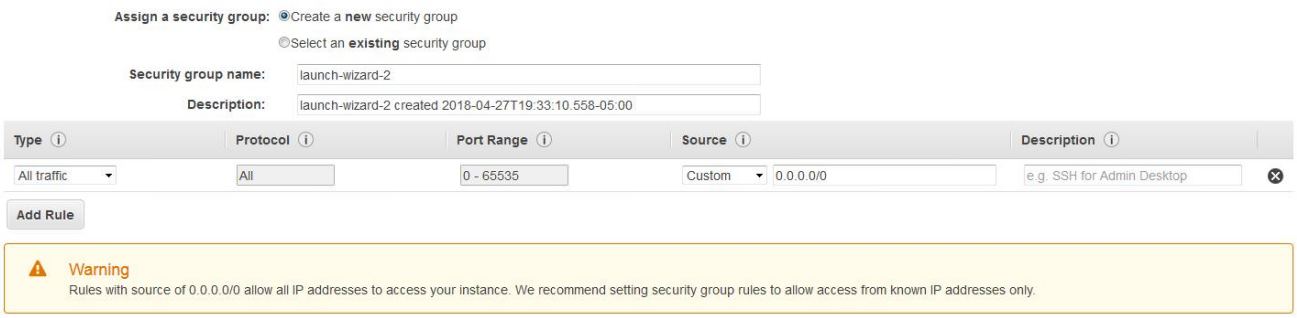
5.- Para la primera parte de la práctica, `t2.micro` es suficiente. (*más adelante cambiaremos este tamaño*). De clic en *Next: Configure Instance Details.*

- Seleccione una sola instancia
- De clic en *Prevention against accidental termination.*

6.- De clic en *Next: Add Storage.* Dejaremos el almacenamiento por default. Para otros casos, aquí puede aumentar el tamaño.

7.- De clic en *Add Tags* y agregue una etiqueta *Key*: `name` y *Value*: `master`

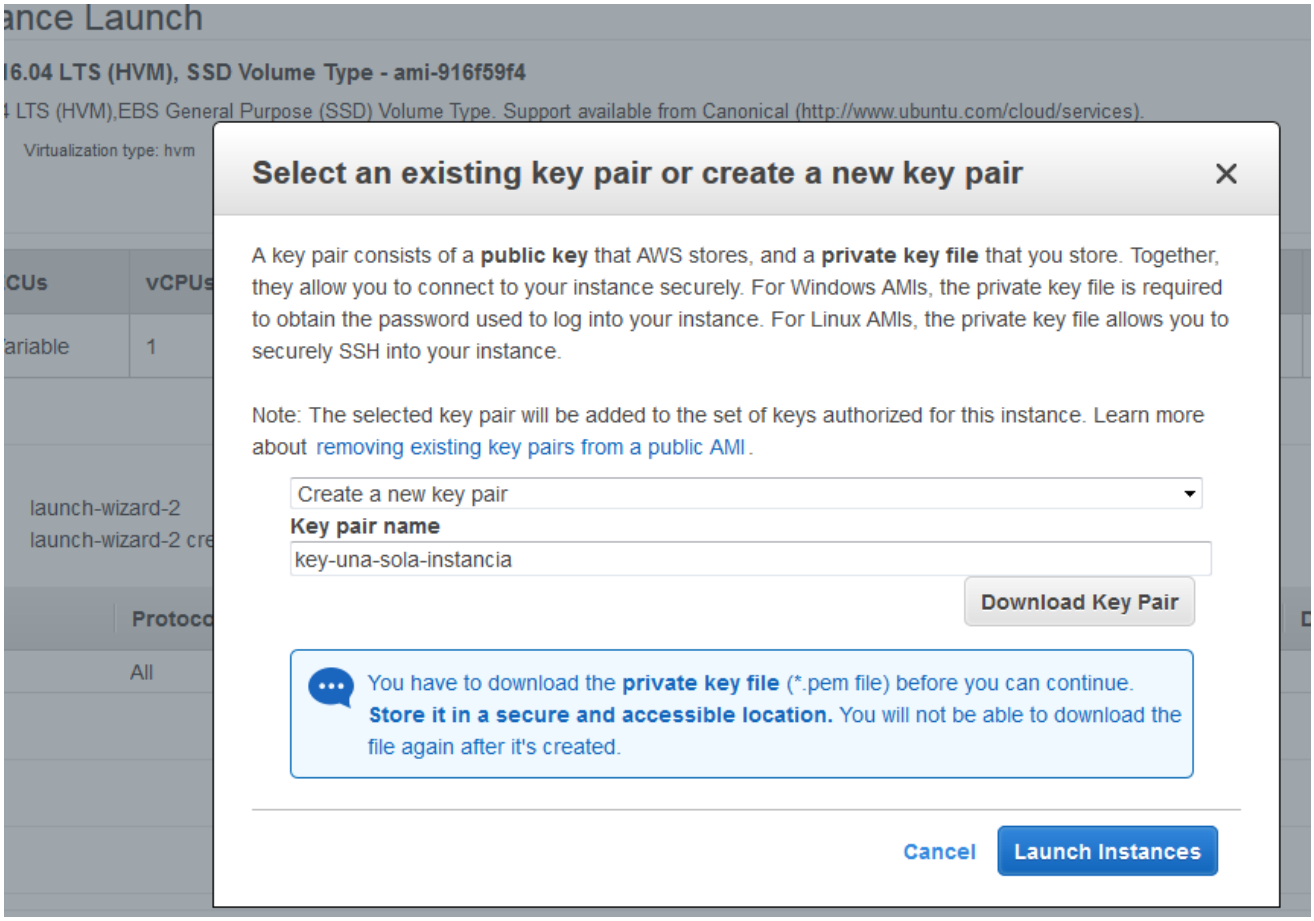
8.- De clic en *Next Configure Security Group*. Por simplicidad, por ahora dejaremos abiertos todos los puertos a todo mundo. Por supuesto, es muy irresponsable trabajar de esta manera tan expuesta en un proyecto real.



9.- De clic en *Review and launch* y si todo está correcto, de clic en *Launch*.

Si es la primera vez que lanza una instancia, se le invitará a generar las llaves PEM necesarias para acceder de forma segura (vía `ssh`) a su instancia.

Seleccione *Create a new key pair*, asígnele un nombre y guárdela en una carpeta.



CUIDADO: Si pierde el archivo que guardó, perderá permanentemente su acceso a la máquina virtual.

De clic en *Launch instances*

10.- De clic en *View Instances.* Aparece una pantalla con las características de la o las instancias que haya lanzado. Selecciónela para ver características con más detalle.

Una muy importante, es el nombre público (*Public DNS*) que se le asignó pues lo estaremos usando intensamente en el resto del tutorial. De clic en el icono con las dos carpetas y guarde el nombre en algún archivo que pueda acceder fácilmente.

En los siguientes comandos, cambie <su_public_DNS> por el nombre que acaba de copiar y grabar.

1.2 Acceder a la instancia

1.2.1.- Linux

Para poder acceder a la instancia con `ssh` en Linux, el archivo donde está la llave PEM debe tener permisos de lectura y escritura únicamente para el dueño.

En los siguientes comandos, cambie <su_archivo_PEM> por el archivo donde almacenó las llaves en el paso **9** anterior.

```
$ sudo chmod 600 <su_archivo_PEM>
$ ssh -i <su_archivo_PEM> ubuntu@<su_public_DNS>

ubuntu@ip-<ip-privada->$
```

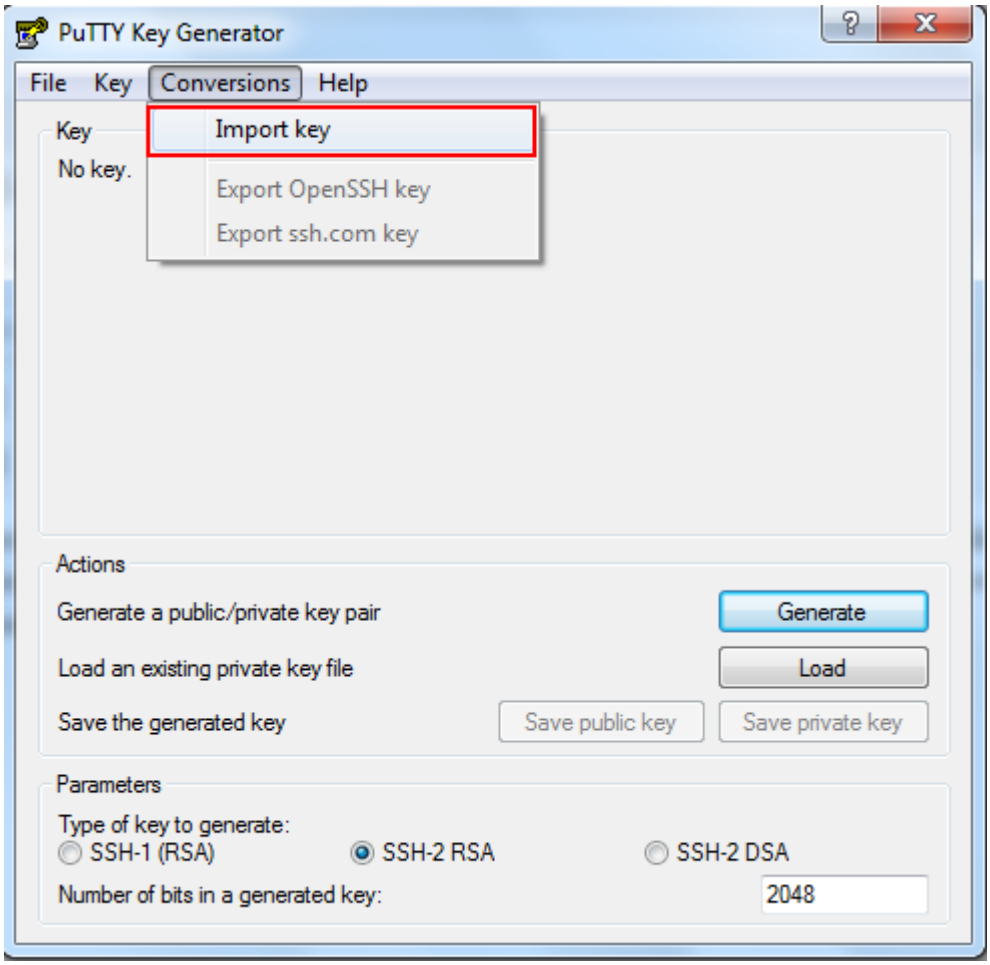
Continúe en la sección **1.3.**

1.2.2.-Windows

Acceder a la VM desde Windows es un poco más complicado porque Windows no tiene implementado el comando `ssh` y la aplicación más popular para hacer conexiones seguras `PuTTY`, no acepta el formato de la llave PEM para hacer la conexión. Por ello, PuTTY también provee una aplicación auxiliar `PuTTYGen`, para generar las llaves compatibles con PuTTY.

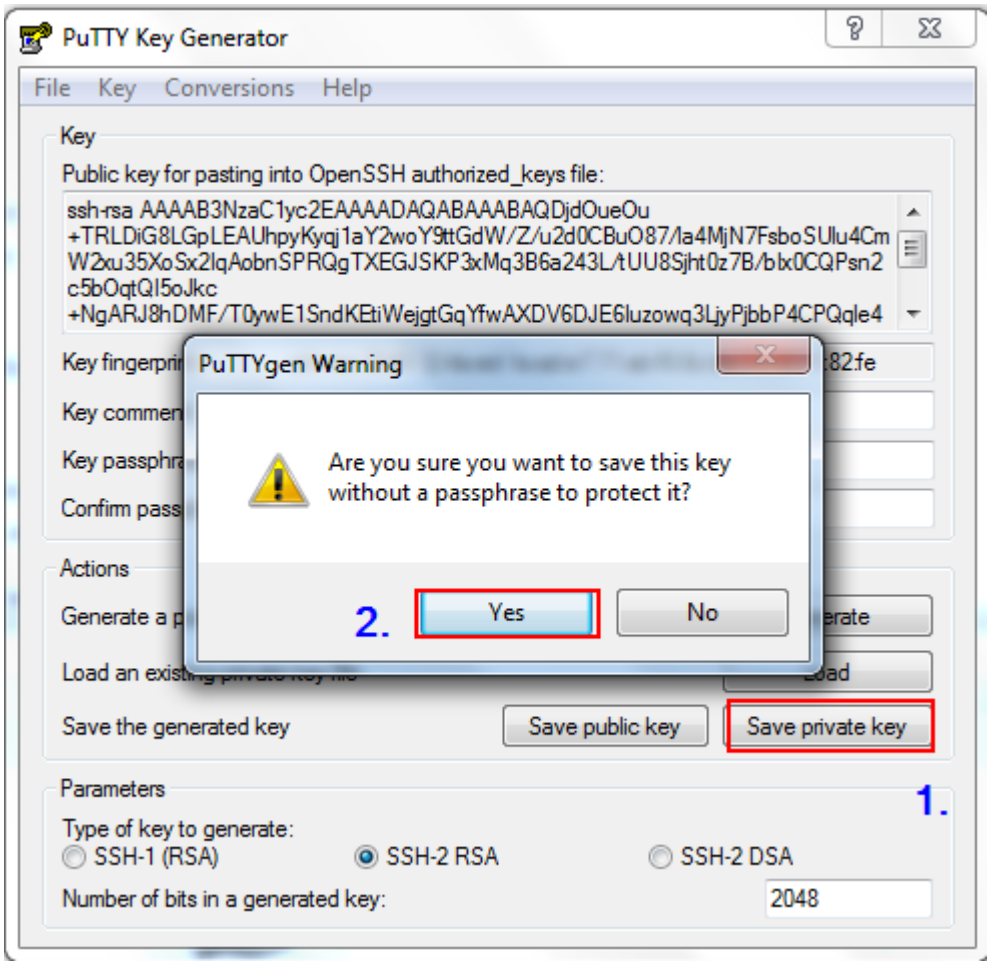
1.- Genere la llave privada.

Lance `PuTTYGen`, de clic en `Conversions` e Importe el archivo con las llaves PEM que se crearon al lanzar la instancia de nuestra VM.



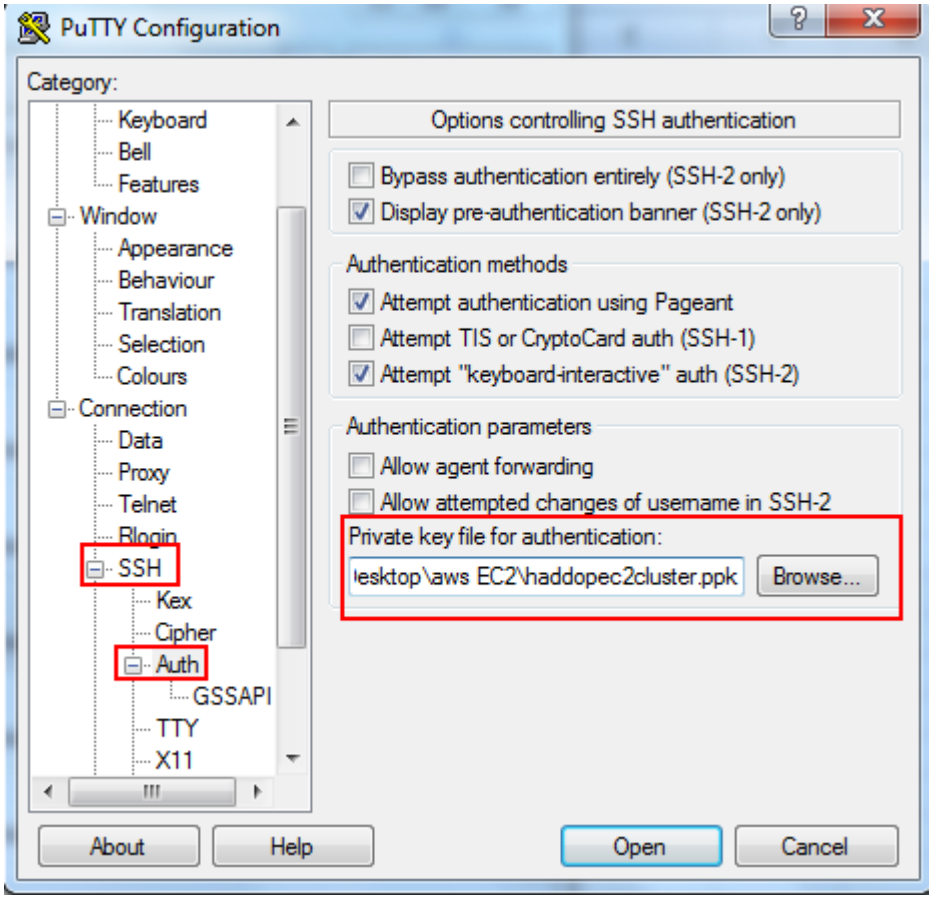
Cargue las llaves, elija la *passphrase* para proteger sus llaves, o deje en blanco esos campos si lo prefiere.

De clic en `Save private key` y guarde la llave *.ppk* en un archivo.

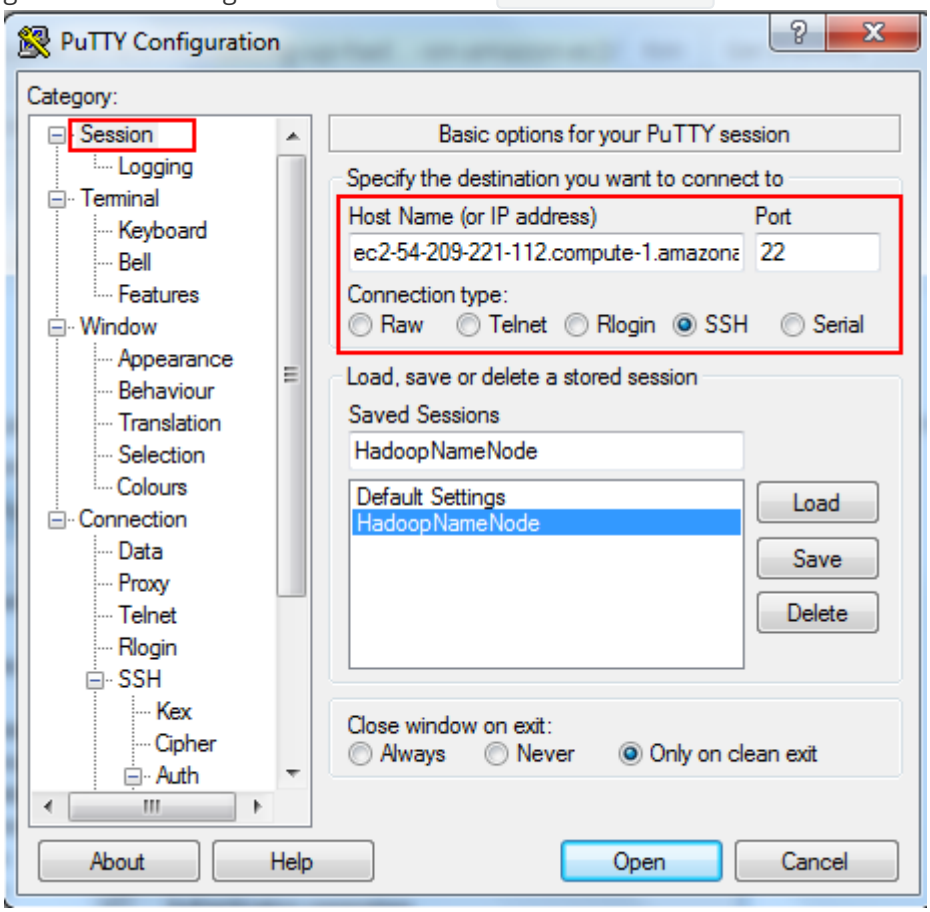


2.- Conexión a la instancia EC2

Lance la aplicación `PuTTY`, de clic en `SSH/Auth` y cargue la llave privada que acaba de guardar.



De clic en `Session` y ponga <su_public_DNS> en la ventana `Host Name`. Para no repetir este paso, puede guardar esta configuración en la sección `Saved Sessions`.



De clic en `Open`. Si despliega un mensaje indicando que la llave no está almacenada, de clic en `Yes`. Se le solicitará un nombre de usuario, ingrese **ubuntu.**

1.3. Instalar Hadoop

Para instalar y ejecutar Hadoop se requiere de Java, que no está instalado en las instancias de AWS. Empecemos por instalar java.

1.3.1.- Para instalar java 8 (puede utilizar una versión anterior) desde el repositorio de Oracle, ejecute los siguientes comandos:

```
$ sudo apt update
$ sudo apt install default-jdk
```

1.3.2.- Para comprobar que Java se instaló correctamente, ejecute el siguiente comando:

```
$ java -version

openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.16.04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
```

1.3.3.- Descargue y descomprima la última versión estable de Hadoop de los espejos de Apache. (Para ver cuál es la última versión estable, consulte esta liga. Nosotros trabajaremos con la versión 2.7.7

```
$ wget https://www-us.apache.org/dist/hadoop/common/hadoop-2.7.7/hadoop-2.7.7.tar.gz

$ sudo tar xzvf hadoop-2.7.7.tar.gz -C /usr/local
$ sudo mv /usr/local/hadoop-2.7.7 /usr/local/hadoop
```

Las banderas del comando `tar` son: *x* para extraer los archivos; *-C dir* para colocar los archivos en el directorio *dir*; *f* para indicar el archivo a extraer; *v* "verbose", muestra avances del comando; *z* descomprime.

1.3.4.- Variables de ambiente

Se deben agregar una serie de variables de ambiente de Java y de Hadoop. La ruta `/usr/bin/java` es en realidad una liga simbólica. Para saber cuál es realmente el "Home Path" de Java, usaremos el comando `readlink` (el comando `sed` es para eliminar el sufijo "bin/java"):

```
$ readlink -f /usr/bin/java | sed "s:bin/java::"
/usr/lib/jvm/java-8-openjdk-amd64/jre
```

Edite el archivo `.bashrc` e inserte las siguientes líneas:

```
$ vim ~/.bashrc

  export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre/
  export HADOOP_HOME=/usr/local/hadoop
  PATH=/usr/local/hadoop/bin:/usr/local/hadoop/sbin:
```

Guarde el archivo modificado. Para que los cambios tengan efecto en la sesión actual, ejecute el siguiente comando:

```
$ source ~/.bashrc
```

Podemos verificar que Hadoop se instaló correctamente y que es accesible desde la variable de ambiente ejecutando el comando:

```
$ hadoop version
Hadoop 2.7.7
Subversion Unknown -r c1aad84bd27cd79c3d1a7dd58202a8c3ee1ed3ac
...
```

1.3.5.- Configuración de Hadoop

Vamos a configurar Hadoop en modo seudo-distribuido para simplificar el despliegue del cluster en la segunda parte. Para que Hadoop funcione adecuadamente, se deben actualizar varios archivos de configuración. Estos se encuentran en `/usr/local/hadoop/etc/hadoop`.

(a) hadoop-env-sh

Debemos remplazar la ubicación del directorio home de Java. De `${JAVA_HOME}` a la ruta que encontró al ejecutar el comando `readlink`. En nuestro caso, la línea queda como:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/jre
```

(b) core-site.xml

Copie y pegue las siguientes líneas:

```
<property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
</property>
```

(c) hdfs-site.xml

En este archivo se especifica el factor de replicación y la ubicación de las carpetas para los datos del NameNode y del DataNode.

Copie y pegue las siguientes líneas:

```
<property>
    <name>dfs.replication</name>
    <value>1</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///usr/local/hadoop/hadoop_data/namenode</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:///usr/local/hadoop/hadoop_data/datanode</value>
</property>
```

<<! Checa si hay que agregar esto

dfs.permission false

```
```

Los directorios especificados no existen. Debemos crearlos y asignarlos al usuario ubuntu:

```
$ mkdir -p $HADOOP_HOME/hadoop_data/datanode
$ mkdir -p $HADOOP_HOME/hadoop_data/namenode
```

1.4 Acceso sin contraseña

En Hadoop, es un requisito poder acceder al ambiente via `ssh` sin contraseñas. Para ello, se generará un par de llaves sin contraseña y la llave pública se almacenará en el archivo de llaves autorizadas:

```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
$ chmod 600 $HOME/.ssh/authorized_keys
```

Para verificar que puede entrar automáticamente (sin contraseña) acceda a localhost.

```
$ ssh localhost
Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-1075-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

# Este es un nuevo shell. Felicidades
# Ahora salimos y regresamos a la sesión anterior:
$ exit
Connection to localhost closed.
$
```

1.5. Ejecución

En esta fase, vamos a ejecutar hadoop/map-reduce como en su primera versión, sin utilizar el gestor de recursos YARN. Esta forma de ejecutar Hadoop ocupa mucho memoria y no genera problemas con el tamaño que elegimos para nuestra instancia en AWS.

Antes de poder trabajar con el sistema de archivos HDFS, debemos formatearlo.

```
$ hdfs namenode -format
```

Estamos listos para iniciar HDFS

```
$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
...
```

Si todo está correcto, debemos ver los siguientes procesos de java:

```
$ jps

9329 Jps
9220 SecondaryNameNode
8887 NameNode
9036 DataNode
```

Ahora debemos crear un directorio en HDFS para nuestro usuario.

```
$ hdfs dfs -mkdir -p /user/ubuntu
```

Verifiquemos que se pueden ejecutar procesos MapReduce de los ejemplos que vienen en la distribución de Hadoop.

1.5.1 Crear directorio en HDFS

Copie los archivos del directorio `$HADOOP_HOME/etc/hadoop` en un directorio `input` en HDFS

```
$ hdfs dfs -mkdir input
$ hdfs dfs -put $HADOOP_HOME/etc/hadoop/*.xml input
$ hdfs dfs -ls
drwxr-xr-x   - ubuntu supergroup          0 2019-05-05 01:27 input
$ hdfs dfs -ls input/ | tail -3
-rw-r--r--   1 ubuntu supergroup       3518 2019-05-05 13:33 input/kms-acls.xml
-rw-r--r--   1 ubuntu supergroup       5540 2019-05-05 13:33 input/kms-site.xml
-rw-r--r--   1 ubuntu supergroup        690 2019-05-05 13:33 input/yarn-site.xml
```

1.5.2 Invocar el jar de ejemplo

```
$ hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-
2.7.7.jar grep input output 'dfs[a-z.]+'
        ...
        Total committed heap usage (bytes)=274939904
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=219
        File Output Format Counters
                Bytes Written=77

$ hdfs dfs -ls
drwxr-xr-x   - ubuntu supergroup          0 2019-05-05 01:27 input
drwxr-xr-x   - ubuntu supergroup          0 2019-05-05 01:32 output

$ hdfs dfs -ls output
--rw-r--r-- 1   ubuntu supergroup   0 2019-05-05 01:32 output/_SUCCESS
--rw-r--r-- 1   ubuntu supergroup   0 2019-05-05 01:32 output/part-r-00000

$ hdfs dfs -cat output/part-r-00000
1       dfsadmin
1       dfs.replication
1       dfs.namenode.name.dir
1       dfs.datanode.data.dir
```

2. El proyecto en MapReduce

En un escenario electoral hipotético, se realizaron encuestas de salida para conocer las preferencias de los electores, así como algunos datos demográficos. De estas encuestas se generó el archivo *votacion.csv* el cual contiene cuatro campos:

1. Hora.- Número entero en el rango [8:17], registra la hora en que se aplicó la encuesta al elector
2. Género.- H = Hombre, M = Mujer, se trata del género del elector
3. Distrito.- Un código que representa el distrito electoral en el que se aplicó la encuesta
4. Candidato.- Número entero en el rango [1:5], representa cada uno de los cinco candidatos que se postularon.

Aunque el archivo *votacion.csv* es muy pequeño, en esta práctica se almacenará en una instancia de Hadoop en su máquina virtual y se realizarán algunos análisis básicos con el modelo MapReduce utilizando guiones (scripts) en Python.

Esta práctica también permitirá experimentar con algunos comandos básicos de Unix/Linux.

2.1. Preparación de datos

En esta práctica trabajaremos exclusivamente a través de la interfaz de la línea de comandos (CLI). Si no lo ha hecho, conéctese a la máquina virtual de su ambiente Hadoop.

2.1.1.- En su directorio `$HOME` cree una carpeta Pr5 con dos subcarpetas: `code` y `data`.

```
$ cd
$ mkdir -p Pr5/code
$ mkdir -p Pr5/data
$ ls -l Pr5
total 8
drxwr-xr-x 2 ubuntu ubuntu 4096 may 21 18:40 code
drxwr-xr-x 2 ubuntu ubuntu 4096 may 21 18:40 data
```

2.1.2.- Descargue los archivos *EjMapper.py* y *EjReducer.py* y guárdelos en la carpeta `code` creada anteriormente. De la misma forma, descargue el archivo *votacion.csv* y guárdelo en la carpeta `data`.

```
$ cd Pr5/code
$ wget https://raw.githubusercontent.com/jincera/Test-Repo/master/EjMapper.py
$ wget https://raw.githubusercontent.com/jincera/Test-Repo/master/EjReducer.py
$ ls
EjMapper.py  EjReducer.py

$ cd ../data
$ wget https://raw.githubusercontent.com/jincera/Test-Repo/master/votacion.csv
$ ls
votacion.csv
```

2.1.3.- Ubique el archivo *votacion.csv* y despliegue las primeras líneas.

```
$ head -3 votacion.csv
12,M,1048,CAND5
15,H,7932,CAND1
13,H,7373,CAND4
```

Como puede observar, se trata de un archivo csv en el que los campos están separados por comas y no tiene encabezado.

Es una buena práctica probar los scripts de Map y Reduce con un conjunto pequeño de datos y, dentro de lo posible, paso a paso desde la línea de comandos, aprovechando los *pipes* de Linux.

2.1.4.- Prepare un archivo de prueba con los primeros 100 registros

```
$ head -100 votacion.csv > vottst.csv
```

Los archivos se encuentran en el sistema de archivos local. Hay que enviarlos a HDFS, el sistema de archivos distribuido de Hadoop:

```
$ hdfs dfs -put votacion.csv
$ hdfs dfs -ls
...
-rw-r--r-- 1 ubuntu supergroup 1591000 2017-05-21 18:59 votacion.csv
...
```

El comando *hdfs dfs* (o el equivalente *hadoop fs* en la versión anterior) indica a Linux que se introducirá una directiva para el sistema de archivos HDFS. Los argumentos siguientes son la directiva y posibles parámetros.

2.2 Programación de scripts

En esta sesión se desarrollarán los scripts con el lenguaje de programación Python. Al ser un lenguaje interpretado, será muy sencillo verificar el comportamiento de los programas map y reduce con pipes de Linux.

Empezaremos por calcular las preferencias electorales para cada uno de los candidatos.

Los archivos *EjMapper.py* y *EjReducer.py* en la carpeta `code`, son ejemplos de un código mapper y de un reducer, respectivamente. El primero lee registros desde la entrada estándar (*stdin*, típicamente el teclado), selecciona dos campos y los imprime en la salida estándar (*stdout*, típicamente la pantalla). Estos dos campos son la tupla < *key,value* > que el reducer tomará para continuar con el procesamiento.

Haga una copia de los archivos `EjMapper.py` y `EjReducer.py` como respaldo en caso de que algo salga mal durante la ejecución de la práctica

```
$ cp EjMapper.py EjMapper.py.bak
$ cp EjReducer.py EjReducer.py.bak
```

2.2.1.- Edite el archivo *EjMapper.py*. Revise el código y modifique la última línea para que se imprima en stdout la columna correspondiente al candidato (key) y un "1" (value). El código del reducer simplemente sumará estas instancias.

2.2.2.- Posiciónese en la carpeta *code* y revise los permisos de los archivos *EjMapper.py* y *EjReducer.py*

```
$ cd ~Pr5/code
$ ls -l
total 8
-rw-r--r-- 1 ubuntu ubuntu 422 may 21 18:44 EjMapper.py
-rw-r--r-- 1 ubuntu ubuntu 819 may 21 18:45 EjReducer.py
```

Para Linux estos archivos no contienen código ejecutable; sólo tienen permisos de lectura y escritura (rw-). Modifique los permisos para que también puedan ser ejecutados:

```
$ chmod 764 *py
hdp>ls -l
total 8
-rwxrw-r- 1 ubuntu ubuntuc 422 may 21 18:44 EjMapper.py
-rwxrw-r- 1 ubuntu ubuntu 819 may 21 18:45 EjReducer.py
```

El comando anterior otorga permisos de lectura, escritura y ejecución (7) al dueño, lectura y escritura (6) a los miembros del grupo y sólo lectura (4) a los demás usuarios.

2.2.3.- Con ayuda del encadenamiento de comandos (pipelining) en Linux, verifique que el código parece funcionar correctamente.

```
$ cat ../data/vottst.csv |./EjMapper.py
...
CAND5   1
CAND4   1
CAND4   1
```

El comando *cat* lee y despliega en pantalla el archivo. El "pipe" (|) toma esa salida y la pasa al siguiente comando, nuestro script *EjMapper.py*, como su propia entrada estándar.

2.2.4.- Abra el archivo *EjReducer.py* y analice su contenido.

Este archivo tiene un acumulador para contar el número de ocurrencias de un operador. Dado que a un proceso *Reduce* llegan los datos ordenados, solo hay que incrementar el acumulador mientras el campo operador (key) no cambie. En principio, este código no debe ser modificado.

2.2.5.- Nuevamente usaremos los pipes de Linux, para verificar que el Reducer parece hacer su función. El comando *sort* en la siguiente instrucción ordena la salida de nuestro mapper, imitando la operación *shuffle* de MapReduce en Hadoop. La salida del sort se toma como entrada para el script *EjReducer.py*.

```
$  ../data/vottst.csv | ./EjMapper.py | sort | ./EjReducer.py
CAND1   8
CAND2   5
CAND3   17
CAND4   48
CAND5   22
```

2.3Ejecución en Hadoop

Ahora que todo parece funcionar correctamente, se puede enviar el código para ser ejecutado en Hadoop. Dado que los programas están escritos en Python, se requiere de la API *hadoop streaming*, la cual permite lanzar tareas MapReduce escritas en prácticamente cualquier lenguaje capaz de recibir datos de la entrada estándar y de escribir resultados en la salida estándar.

2.3.1.- Desde la terminal, ejecute el siguiente comando:

```
$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.7.jar -files
EjMapper.py,EjReducer.py -input votacion.csv -output OpElec -mapper EjMapper.py -
reducer EjReducer.py

$ hadoop dfs -ls OpElec

Found 2 items
-rw-r--r--   1 ubuntu supergroup          0 2018-04-28 23:49 05/_SUCCESS
-rw-r--r--   1 ubuntu supergroup         64 2018-04-28 23:49 05/part-00000

$ hdfs dfs -cat OpElec/part-00000

CAND1   10056
CAND2   9884
CAND3   15051
CAND4   40018
CAND5   24991
```

Los argumentos de la instrucción anterior son:

- files: Los archivos adicionales que se deben enviar al cluster HDFS; en nuestro caso, los scripts de map y reduce. Debe ser la primer opción en la línea de comandos.
- input: El archivo de donde se leerán los datos que se envían a los procesos Map
- output: El directorio donde se almacenan los resultados (o los mensajes de error)
- mapper, reducer: Los scripts con los códigos para los procesos Map y Reduce

Como vamos a estar utilizando frecuentemente esta API, quizás le gustaría crear una variable de ambiente con la ruta del archivo jar para simplificar la escritura del comando:

```
$ export STRJAR=/usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.7.jar
```

De esta manera, el comando anterior se invocaría así (**Debemos cambiar el directorio de salida. Hadoop no permite reescribir archivos**):

```
$ hadoop jar $STRJAR -files EjMapper.py,EjReducer.py -input votacion.csv -output
OpElec2 ...
```

Revise brevemente la salida, compruebe que no hubo errores, identifique cuántas tareas map y reduce se dispararon.

2.3.2.- Verifique que el resultado se generó correctamente:

```
$ hadoop fs -cat OpcionElectoral/part-00000
CAND1   10056
CAND2   9884
CAND3   15051
CAND4   40018
CAND5   24991
```

Así se nombran los archivos de salida en la primera versión de MapReduce. En la versión más reciente se nombran part-[m|r]-xxxxx para indicar si el archivo se generó a la salida de un Mapper (m) o de un Reducer (r). El número (xxxxx) es un identificador único en esa carpeta para distinguir entre los resultados de (potencialmente) muchas tareas.

2.3.3.- Ahora lance el job con la API de Hadoop streaming y el archivo votación. **Aprovecharemos para mostrar cómo se invoca la ejecución de varias tareas Reducer, lo que puede ser útil para procesar grandes volúmenes de datos**.

```
$ hadoop jar $STRJAR -files EjMapper.py,EjReducer.py -input votacion.csv -output
OpElect3
-mapper EjMapper.py -reducer EjReducer.py -numReduceTasks 2
```

Verifique que obtuvo los mismos resultados, solo que en dos archivos de salida:

```
$ hdfs dfs -ls OpElec3
-rw-r--r--  1 root hdfs   0 2017-05-21  22:48 OpElec3/_SUCCESS
-rw-r--r--  1 root hdfs   0 2017-05-21  22:48 OpElec3/part-00000
-rw-r--r--  1 root hdfs   0 2017-05-21  22:48 OpElec3/part-00001

$ hdfs dfs -cat OpElec3/part-00000
CAND1  10056
CAND3  15051
CAND5  24991

$ hdfs dfs -cat OpElec3/part-00001
CAND2  9884
CAND4  40018
```

El ambiente decidió por sí mismo cómo distribuir las llaves entre los dos Reducers. De lo que podemos tener certeza, es que todos los registros con la misma llave, llegaron al mismo Reducer.

Para terminar correctamente los procesos de Hadoop, ejecute el siguiente comando:

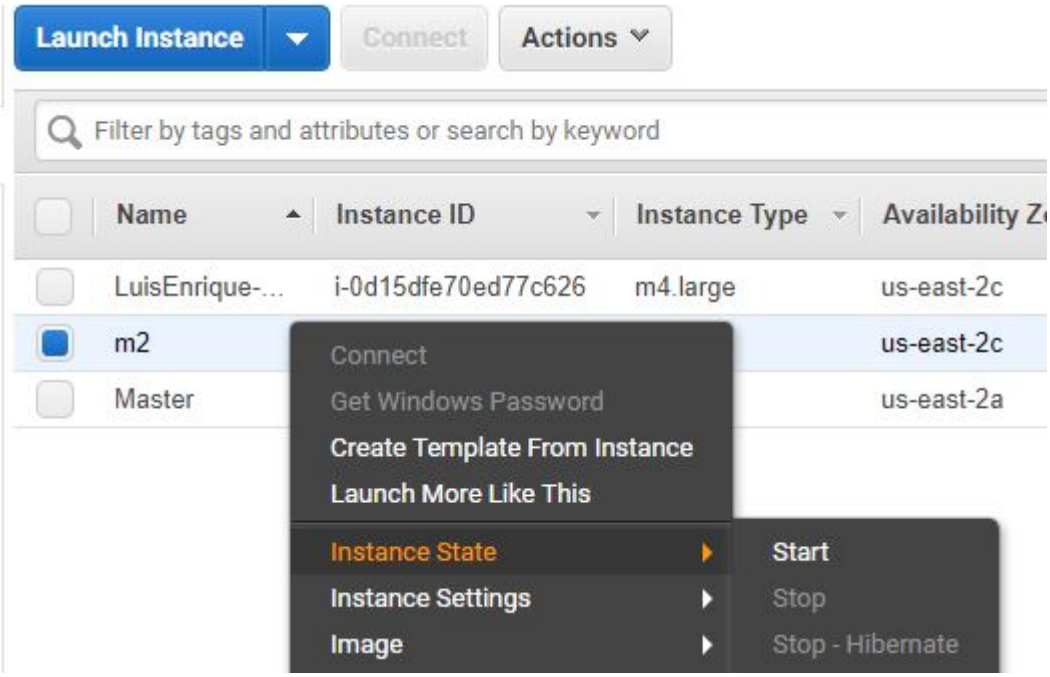
```
$ stop-dfs.sh
```

3. Preparación para el cluster

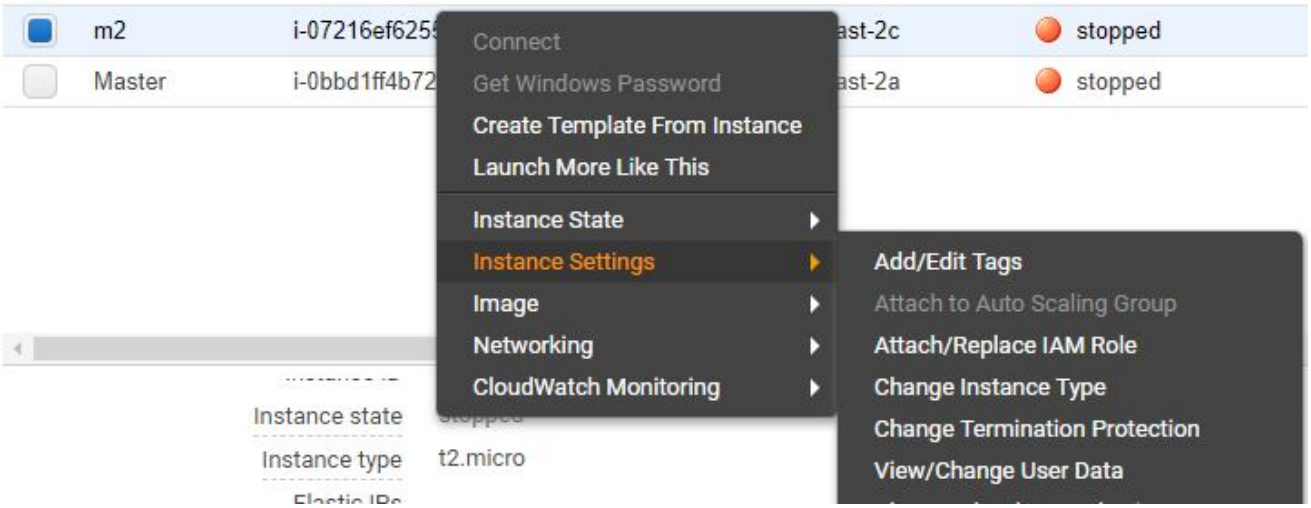
El cluster de Hadoop se implementa con contenedores y el gestor de recursos YARN. La instancia t2.micro, con 1GB de memoria, es demasiado pequeña para soportar los requerimientos de memoria de Hadoop 2.0. Debemos cambiarla a una instancia t2.large

3.1. Modificación de la instancia en AWS

En la consola de administración de AWS, asegúrese que la instancia está detenida. Para ello, de clic derecho en la instancia, seleccione `Instance State/Stop`



Una vez detenida, de clic derecho en `Instance Settings/Change Instance Type` y seleccione `t2.large` y de clic en `Apply`.



Posteriormente, inicie la instancia y conéctese nuevamente a ella. *Observe que la dirección IP pública de la instancia ha cambiado*.

3.2 Configuración de archivos

Para YARN, se deberán configurar los siguientes archivos (recuerde que se encuentran en la carpeta `/usr/local/hadoop/etc/hadoop`:

3.2.1 mapred-site.xml

Copie el archivo `mapred-site.xml.template` en `mapred-site.xml y edite el nuevo archivo para agregar las líneas de configuración siguientes:

```
$ cp mapred-site.xml.template mapred-site.xml
$ vim mapred-site.xml


<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

3.2.2 yarn-site.xml

Edite el archivo para incluir las siguientes líneas de configuración:

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

3.3 Ejecución

Empecemos por lanzar los procesos de mapreduce y verificar que los archivos y carpetas en HDFS todávia se encuentran ahí (aunque, de hecho, al lanzar el cluster, deberemos reformatear el sistema de archivo).

```
$ start-dfs.sh
Starting namenodes on [localhost]
...
$ hdfs dfs -ls
Found 5 items
drwxr-xr-x   - ubuntu supergroup          0 2019-05-05 13:44 OpElec
drwxr-xr-x   - ubuntu supergroup          0 2019-05-05 13:51 OpElec2
drwxr-xr-x   - ubuntu supergroup          0 2019-05-05 13:43 input
drwxr-xr-x   - ubuntu supergroup          0 2019-05-05 13:36 output
-rw-r--r--   1 ubuntu supergroup    1591000 2019-05-05 13:40 votacion.csv

$ jps
2178 Jps
1827 DataNode
1668 NameNode
2024 SecondaryNameNode
ubuntu@ip-172-31-46-214:~/pr5/code$
```

Ahora lanzamos los procesos de YARN y verificamos que todos se estén ejecutando:

```
$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/logs/yarn-ubuntu-
resourcemanager-ip-172-31-46-214.out
localhost: starting nodemanager, logging to /usr/local/hadoop/logs/yarn-ubuntu-
nodemanager-ip-172-31-46-214.out

$ jps
2241 ResourceManager
1827 DataNode
1668 NameNode
2629 Jps
2024 SecondaryNameNode
2380 NodeManager
```

Los procesos `ResourceManager` y `NodeManager` están levantados.

3.3.1 Tareas MapReduce en YARN

Vamos a lanzar el mismo jar hadoop-streaming de la sección anterior (no olvide de cambiar la carpeta destino) pero en esta ocasión Hadoop está configurado para trabajar con YARN. Verá que el lanzamiento y la ejecución de tareas (para este caso tan pequeño) parecen mucho más lentos. Lo que ocurre, es que tenemos todos los procesos ejecutándose en una sola máquina virtual.

```
...
-46-214:8088/proxy/application_1557101849692_0002/
19/05/06 00:22:48 INFO mapreduce.Job: Running job: job_1557101849692_0002
19/05/06 00:23:25 INFO mapreduce.Job: Job job_1557101849692_0002 running in uber mode :
false
19/05/06 00:23:25 INFO mapreduce.Job:  map 0% reduce 0%
19/05/06 00:23:31 INFO mapreduce.Job:  map 100% reduce 0%
19/05/06 00:23:36 INFO mapreduce.Job:  map 100% reduce 100%
...
              WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=1595096
        File Output Format Counters
                Bytes Written=20
19/05/06 00:23:37 INFO streaming.StreamJob: Output directory: OpElec4
```

Verifique que las tareas se ejecutaron y los resultados se almacenaron en HDFS:

```
$ hdfs dfs -ls OpElec4
-rw-r--r--  1 root hdfs   0 2017-05-21  22:48 OpElec4/_SUCCESS
-rw-r--r--  1 root hdfs   0 2017-05-21  22:48 OpElec4/part-00000
-rw-r--r--  1 root hdfs   0 2017-05-21  22:48 OpElec4/part-00001

$ hdfs dfs -cat OpElec4/part-00000
CAND1  10056
CAND2  9884
CAND3  15051
CAND4  40018
CAND5  24991
```