1: Introduction

Link: <https://www.youtube.com/watch?v=efzIn_CtvAM>

Notes:

**Lesson Overview**

By the end of the lesson, you will be able to:

* Distinguish between setting up a Spark Cluster using both Local and Standalone Mode
* Set up Spark Cluster in AWS
* Use Spark UI
* Use AWS CLI
* Create EMR using AWS CLI
* Create EMR Cluster
* Test Port Forwarding
* Use Notebooks on your Spark Cluster
* Write Spark Scripts
* Store and Retrieve data on the Cloud
* Read and Write to Amazon S3
* Understand the distinction between HDFS and S3
* Reading and Writing Data to HDFS

NEXT

2: From local to standalone mode

Link: <https://www.youtube.com/watch?v=EeBWbABm_Qc>

Notes:

### Overview of the Set up of a Spark Cluster

1. **Amazon S3** will store the dataset.
2. We rent a cluster of machines, i.e., our **Spark Cluster**, and iti s located in AWS data centers. We rent these using AWS service called **Elastic Compute Cloud (EC2)**.
3. We log in from your local computer to this Spark cluster.
4. Upon running our Spark code, the cluster will load the dataset from **Amazon S3** into the cluster’s memory distributed across each machine in the cluster.

#### New Terms:

* **Local mode**: You are running a Spark program on your laptop like a single machine.
* **Standalone mode**: You are defining Spark Primary and Secondary to work on your (virtual) machine. You can do this on EMR or your machine. Standalone mode uses a resource manager like YARN or Mesos.

## Local vs Standalone

In your own words, please describe key differences between the local and standalone modes of Spark.

### Your reflection

**Local mode: Local machine is the only resource, e.g, cpu, ram, and sdd standalone mode: in this mode we use spark capability more better than local mode. We use distributed computing.**

### Things to think about

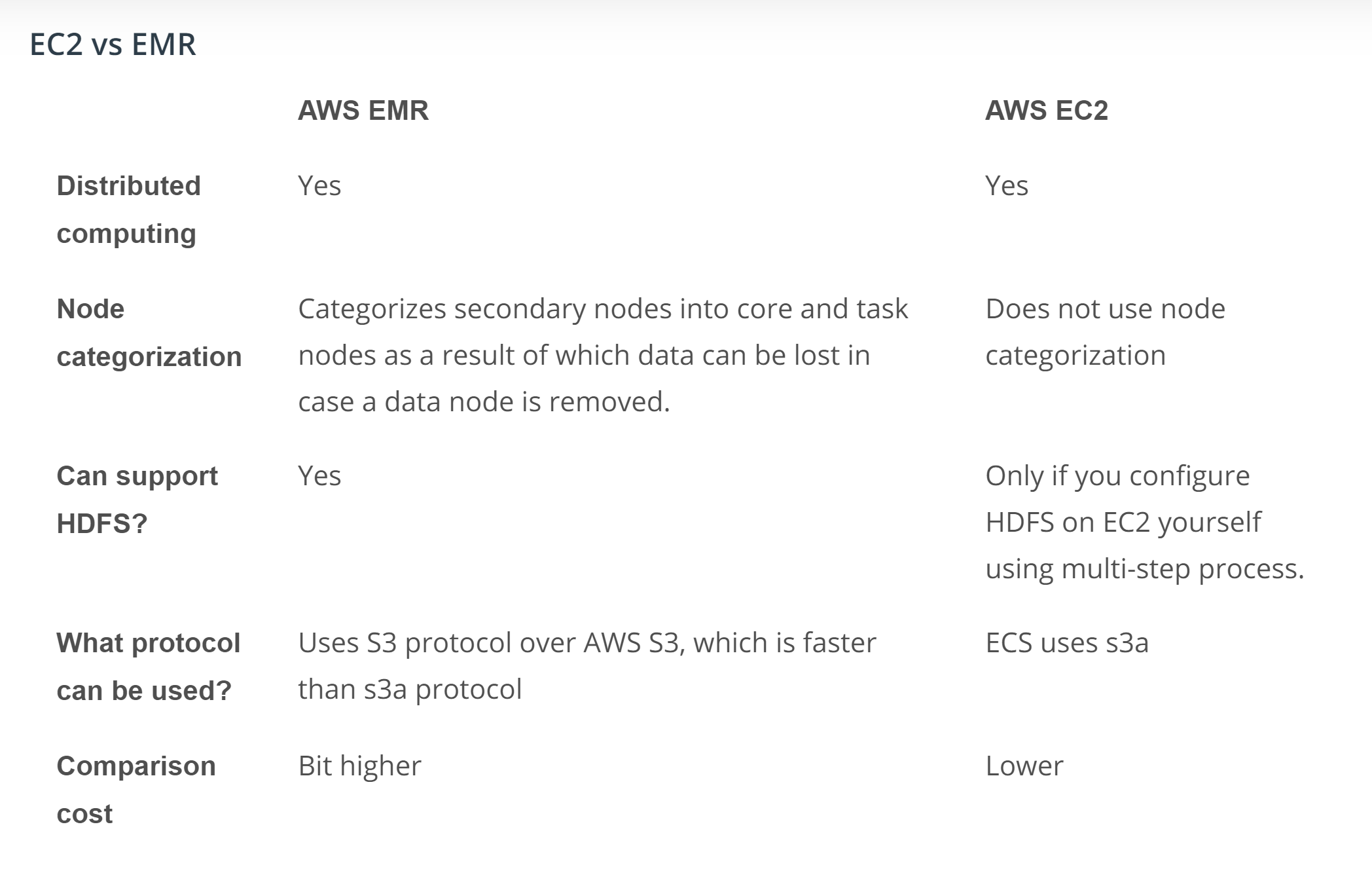
Thanks for your response. Here is what I would’ve said for this question:

Local mode means Spark is running on your local machine. Standalone mode is distributed and uses resource management like Yarn or Mesos.

3: Setup instructions AWS

Link: <https://www.youtube.com/watch?v=ZVdAEMGDFdo>

Notes:



### Circling back about HDFS

Previously we have looked over the Hadoop Ecosystem. To refresh those concepts, we have provided reference material here. HDFS (Hadoop Distributed File System) is the file system. HDFS uses MapReduce system as a resource manager.

Spark can replace the MapReduce algorithm. Since Spark does not have its own distributed storage system, it leverages using HDFS or AWS S3, or any other distributed storage. Primarily in this course, we will be using AWS S3, but let’s review the advantages of using HDFS over AWS S3.

### What is HDFS?

HDFS (Hadoop Distributed File System) is the file system in the Hadoop ecosystem. Hadoop and Spark are two frameworks providing tools for carrying out big-data related tasks. While Spark is faster than Hadoop, Spark has one drawback. It lacks a distributed storage system. In other words, Spark lacks a system to organize, store and process data files.

### MapReduce System

HDFS uses MapReduce system as a resource manager to allow the distribution of the files across the hard drives within the cluster. Think of it as the MapReduce System storing the data back on the hard drives after completing all the tasks.

Spark, on the other hand, runs the operations and holds the data in the RAM memory rather than the hard drives used by HDFS. Since Spark lacks a file distribution system to organize, store and process data files, Spark tools are often installed on Hadoop because Spark can then use the Hadoop Distributed File System (HDFS).

Watch VIDEO

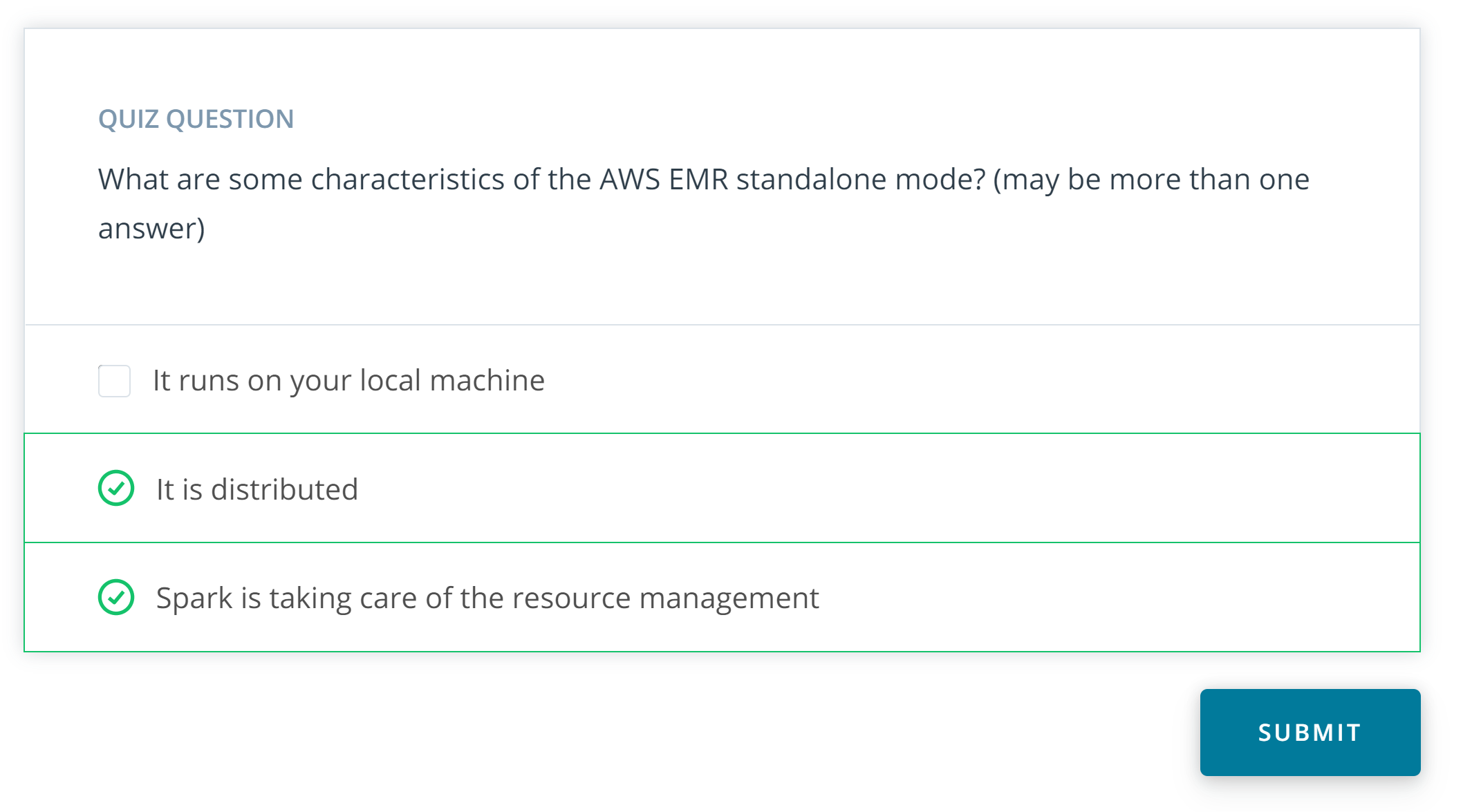
### Why do you need **EMR Cluster**?

Since a Spark cluster includes multiple machines, in order to use Spark code on each machine, we would need to download and install Spark and its dependencies. This is a manual process. **Elastic Map Reduce** is a service offered by AWS that negates the need for you, the user, to go through the manual process of installing Spark and its dependencies for each machine.

### Setting up AWS

Please refer to the latest [**AWS documentation to set up an EMR Cluster**](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs-launch-sample-cluster.html).

Let’s pause to do a quick check for understanding from a previous page.



4: Install and configure CLI v2

Link:

Notes:

The AWS Command Line Interface (AWS CLI) is a command-line tool that allows you to interact with AWS services using commands in your terminal/command prompt.

AWS CLI enables you to run commands to provision, configure, list, delete resources in the AWS cloud. Before you run any of the [**aws commands**](https://docs.aws.amazon.com/cli/latest/reference/), you need to follow two steps:

1. Install CLI
2. Configure CLI

## 1. Install AWS CLI v2

Refer to the official [**AWS instructions to install/update AWS CLI**](https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html) (version 2) based on your underlying OS. You can verify the installation using the following command in your terminal (macOS)/cmd (Windows).

*# Display the folder that contains the symlink to the aws cli tool*

which aws

*# See the current version*

aws *--version*

See the sample output below. Note that the exact version of AWS CLI and Python may vary in your system.



Verify the successful installation of AWS CLI 2

## 2. Configure CLI

You will need to configure the following four items on your local machine before you can interact with any of the AWS services:

* **Access key** - It is a combination of Access Key ID and a Secret Access Key. Together, they are referred to as Access key. You can generate an Access key from the AWS IAM service, and specify the level of permissions (authorization) with the help of IAM Roles.
* **Default AWS Region** - It specifies the AWS Region where you want to send your requests by default.
* **Default output format** - It specifies how the results are formatted. It can either be a json, yaml, text, or a table.
* **Profile** - A collection of settings is called a profile. The default profile name is default, however, you can create a new profile using the aws configure --profile new\_name command. A sample command is given below.

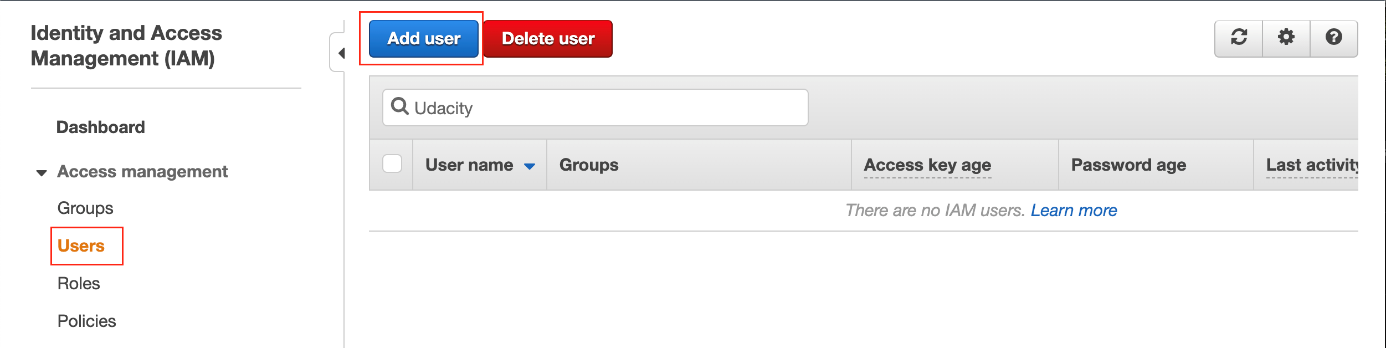
Let’s create an Access key using [**AWS IAM**](https://docs.aws.amazon.com/IAM/latest/UserGuide/intro-structure.html), in order to allow CLI to use AWS services.

### 2.1. Create an IAM User and copy the access key

AWS Identity and Access Management (IAM) service allows you to authorize users / applications (such as AWS CLI) to access AWS resources.

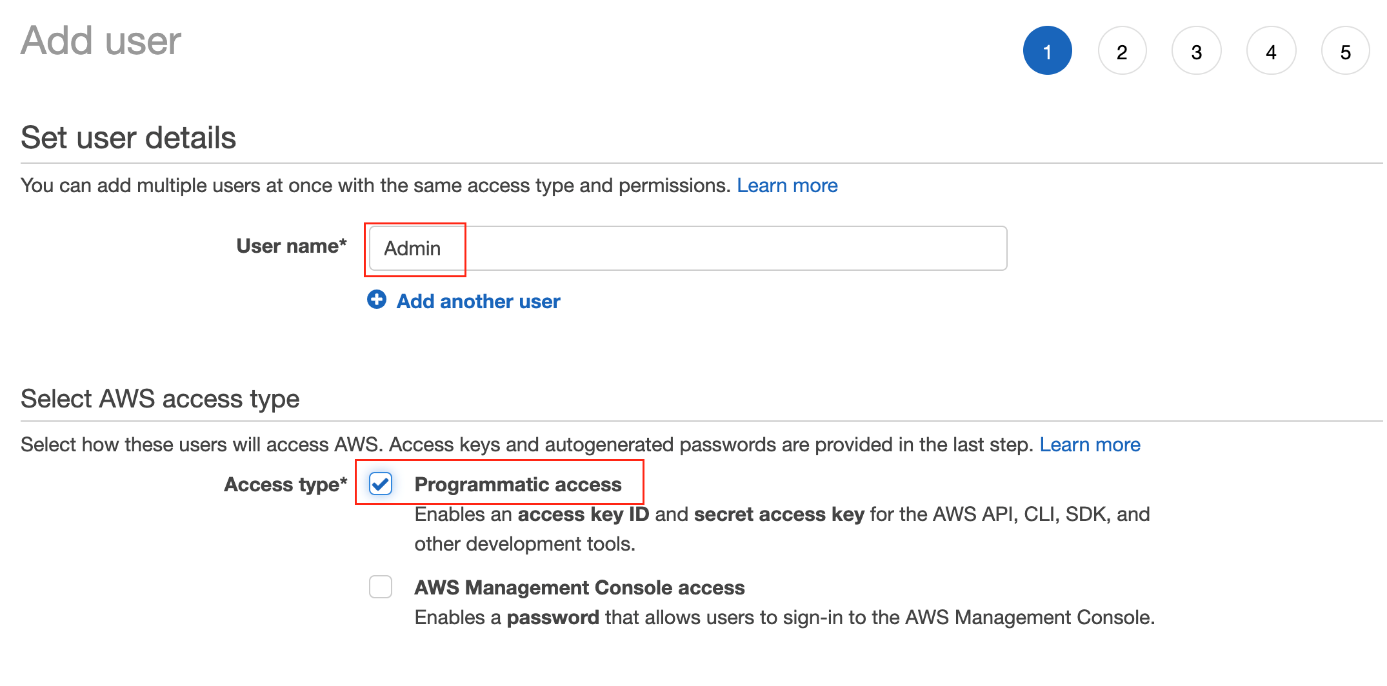
The access key is a .csv file that contains an **Access Key ID** and a **Secret Access Key.**Let's see the steps to download an Access key (.csv) and use it later.

* Navigate to the [**IAM Dashboard**](https://console.aws.amazon.com/iam/home#/home), and create an IAM user.



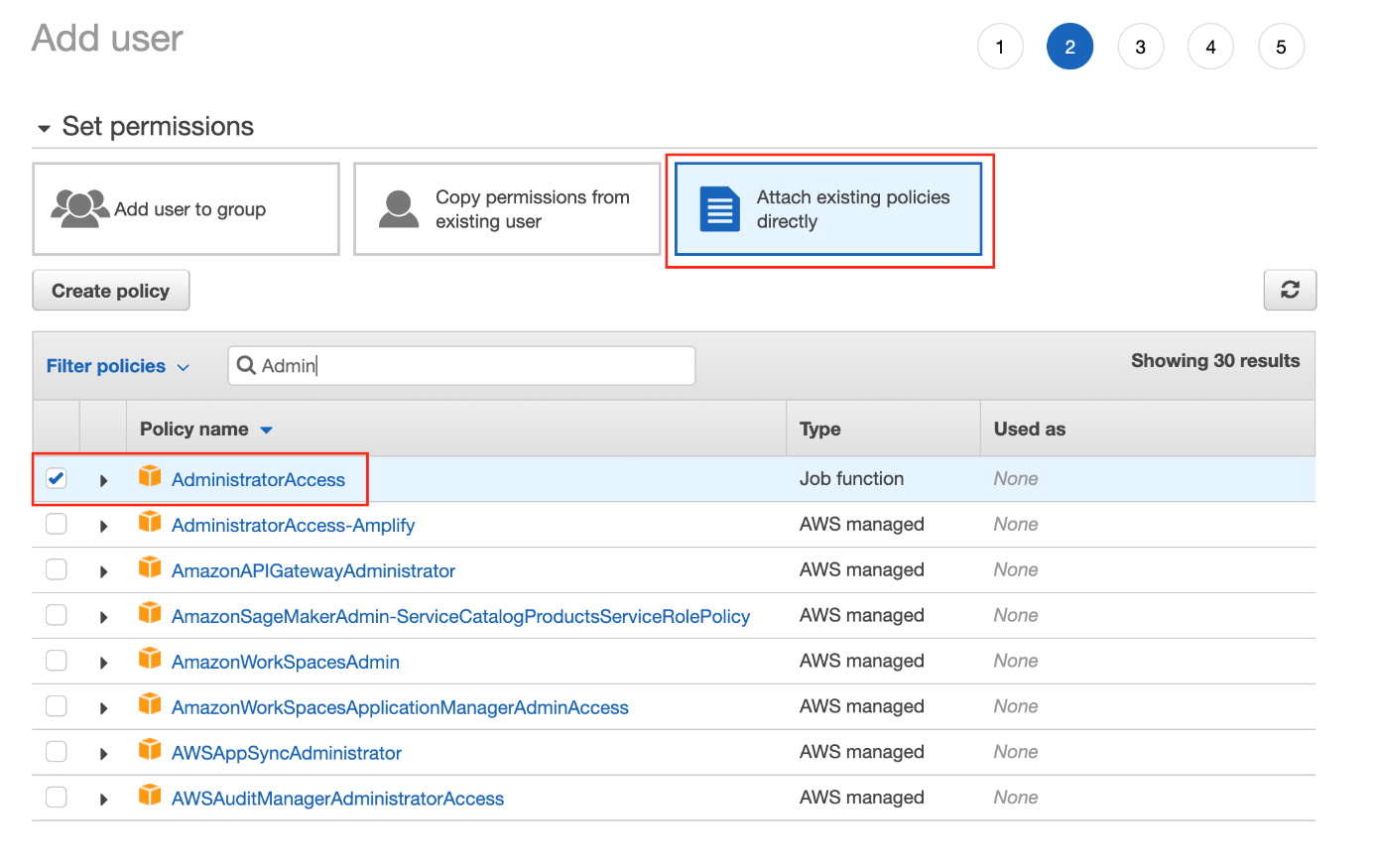
Add a new user

* Set the user details, such as the name, and access type as Programmatic access only.



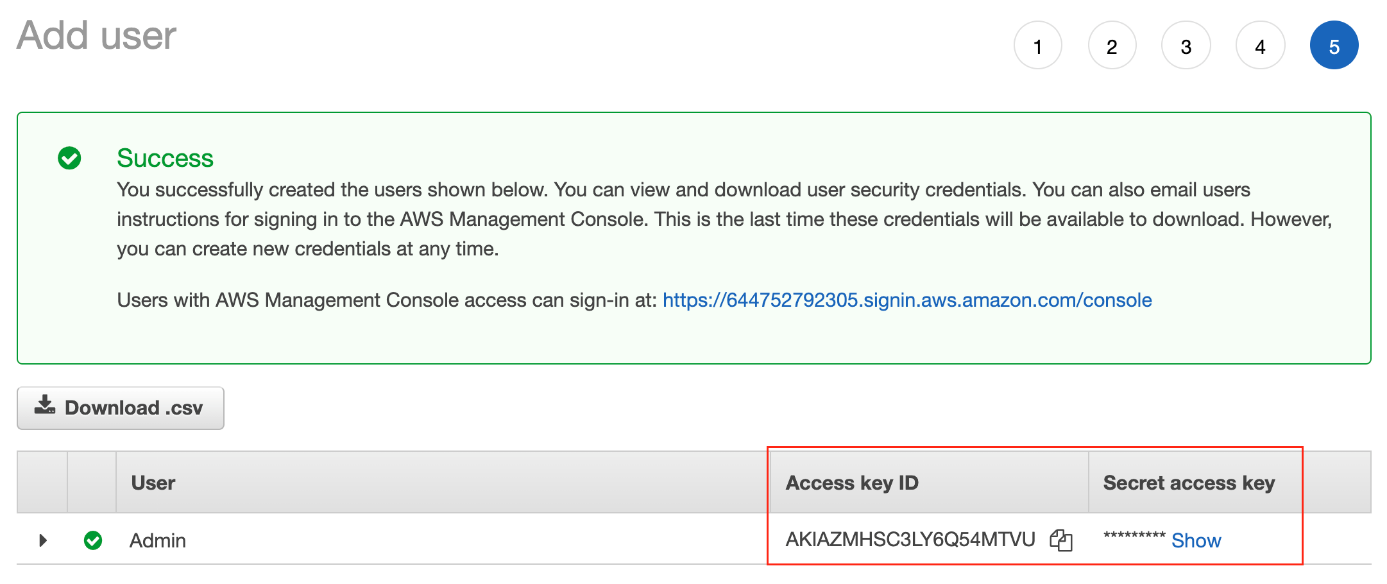
Set the user name, and type (mode) of access

* Set the permissions to the new user by attaching the AWS Managed **AdministratorAccess** policy from the list of existing policies.



Attach the AdministratorAccess policy from the list of pre-created policies

* Provide tags [optional], review the details of the new user, and finally create the new user.
* After a user is created successfully, download the access key file (.csv) containing the Access Key ID and a Secret Access Key. You can even copy the keys and stay on the same page. **Don’t skip this step as this will be your only opportunity to download the secret access key file.**



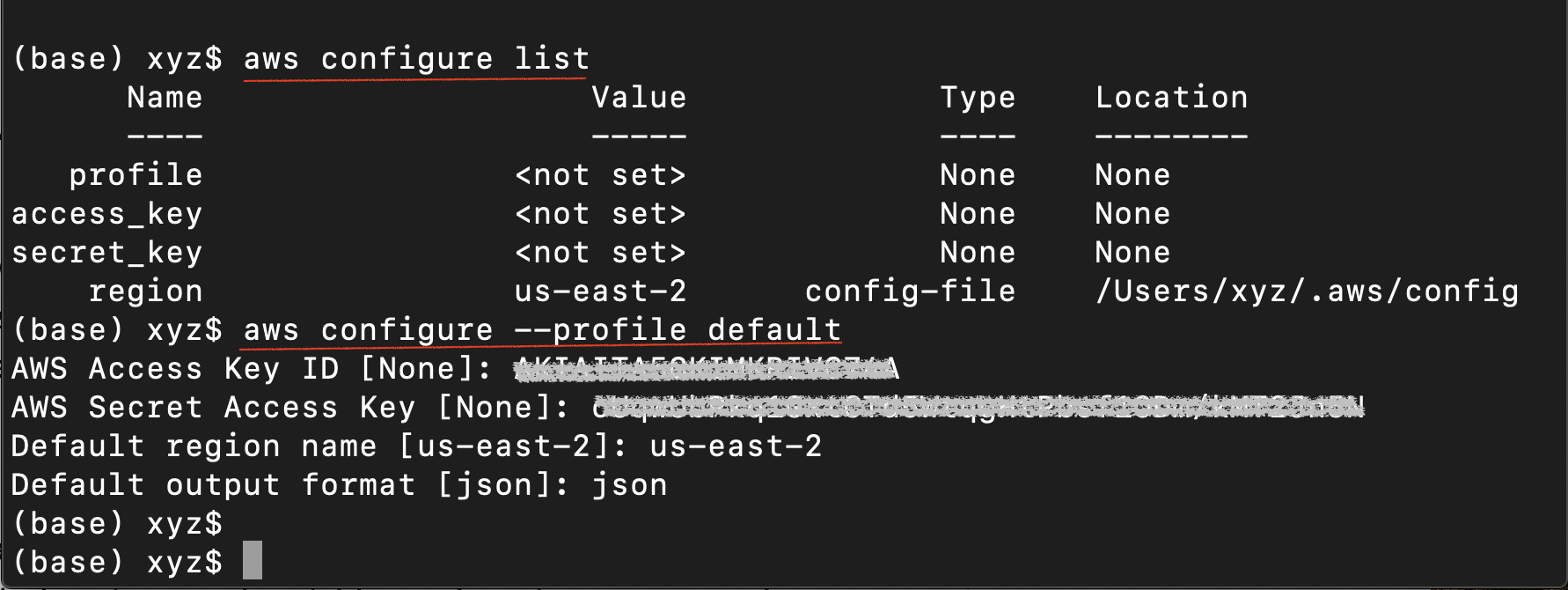
Copy the Access key of the new user

### 2.2. Configure the credentials and the environment variables

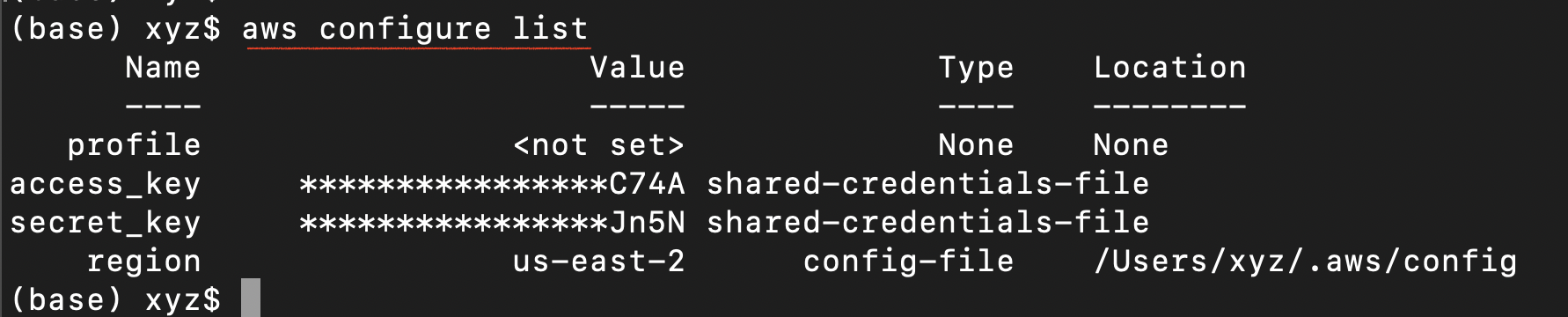
* If you have closed the web console that showed the access key, you can open the downloaded access key file (.csv) to copy the keys later. It should be something similar to:
* AWSAccessKeyId=WANI9WATIG63GKCXA89VC74A
* AWSSecretKey=kMT2Jn5NPkq1GxtoUqwUbgHtPbsf1ODm/Pbsf1OD
* Navigate to the home directory and check the current configuration:
* cd
* aws configure list
* Store the access key in a default file ~/.aws/credentials and store the profile in the ~/.aws/config file using the following command.
* *# Set the default profile credentials in the "credentials" file*
* aws configure --profile default

Upon prompt, paste the copied access key (access key id and secret access key). Enter the default region as us-east-1 and output format as json.

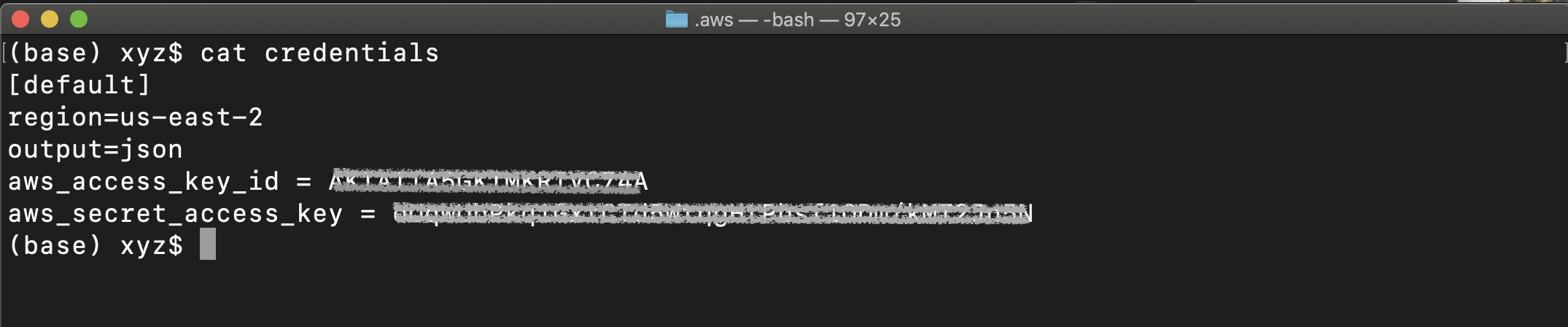
* Let the system know that your sensitive information is residing in the .aws folder
* export AWS\_CONFIG\_FILE=~/.aws/config
* export AWS\_SHARED\_CREDENTIALS\_FILE=~/.aws/credentials



List your present configuration, and then configure your default aws profile



A successful configuration



View the credentials file using cat ~/.aws/credentials command

* Check the successful configuration of the AWS CLI, by running an AWS command:
* aws iam **list**-users

The output will display the details of the recently created user:

{

"Users": [

{

"Path": "/",

"UserName": "Admin",

"UserId": "AIDAZMXYZ3LY2BNC5ZM5E",

"Arn": "arn:aws:iam::388752792305:user/Admin",

"CreateDate": "2021-01-28T13:44:15+00:00"

}

]

}

**Note for Windows users:** If you are facing issues in following the commands above, refer to the detailed instructions here -

1. [**Configuration basics**](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html)
2. [**Configuration and credential file settings**](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html)
3. [**Environment variables to configure the AWS CLI**](https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html)

After a successful credential set-up, your "credentials" file will look like:

5: AWS CLI – Create EMR Cluster

Link:

Notes:

Let's learn how to create an EMR cluster from the CLI, and configure the related settings.

## 1. aws emr create-cluster command

While creating EMR through AWS console has been shown, but if you know your instances' specificity, such as which applications you need or what kind of clusters you’ll need, you can reuse the aws emr create-cluster command below multiple times.

aws emr **create**-cluster *--name <cluster\_name> \*

*--use-default-roles --release-label emr-5.28.0 \*

*--instance-count 3 --applications Name=Spark Name=Zeppelin \*

*--bootstrap-actions Path="s3://bootstrap.sh" \*

*--ec2-attributes KeyName=<your permission key name> \*

*--instance-type m5.xlarge --log-uri s3:///emrlogs/*

1. **Options of the aws emr create-cluster command** - Let’s break down the command and go over each option to know its responsibility.
   * --name : You can give any name of your choice. This will show up on your AWS EMR UI.
   * --release-label: This is the version of EMR you’d like to use.
   * --instance-count: Annotates instance count. One is for the primary, and the rest are for the secondary. For example, if --instance-count is given 4, then 1 instance will be reserved for primary, then 3 will be reserved for secondary instances.
   * --applications: List of applications you want to pre-install on your EMR at the launch time
   * --bootstrap-actions: The Path attribute provides the path to a file (residing in S3 or locally) that contains a script that runs during a bootstrap action. The script may set environmental variables in all the instances of the cluster. This file must be accessible to each instance in the cluster.
   * --ec2-attributes KeyName: Specify your permission key name, for example, if it is MyKey.pem, just specify MyKey for this field
   * --instance-type: Specify the type of instances you want to use. [**Detailed list can be accessed here**](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-supported-instance-types.html), but find the one that can fit your data and your budget.
   * --log-uri: S3 location to store your EMR logs in. This log can store EMR metrics and also the metrics/logs for submission of your code.
2. **Reference** - You can refer to an even more detailed explanation about all possible options of the aws emr create-cluster command at [**CLI command reference**](https://awscli.amazonaws.com/v2/documentation/api/latest/reference/emr/create-cluster.html).

## 2. Exercise: Create an EMR cluster using AWS CLI

Follow the instructions given below:

### 2.1. Prerequisite

1. **AWS CLI** - Install AWS CLI on your local computer. Refer to the [**AWS instructions to install/update AWS CLI**](https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html) (version 2) based on your underlying OS.
2. **Set up Access credentials using AWS IAM** - Generate and save a new Access key (access key ID, and a secret key) locally in your system, which will allow your CLI to create an EMR cluster. You will have to configure the environment variables so that the aws configure command can run properly.
3. **EC2 Login Key-Pair** - You should have an EC2 login key-pair to access your EC2 instances in the cluster. You can generate a key-pair from the [**EC2 dashboard**](https://console.aws.amazon.com/ec2/v2/home). Remember, a key-pair is a pair of (encrypted) public and (unencrypted PEM encoded) private keys. The public key is placed automatically on the instance, and the private key is made available to the user, just once. Suppose, your private key file name is AWS\_EC2\_Demo.pem, then you should use only "AWS\_EC2\_Demo" in the script below, with the option --ec2-attributes.

### 2.2. Create an EMR Cluster

1. **Create default roles in IAM** - Before you run the aws emr create-cluster command, make sure to have the necessary roles created in your account. Use the following command.
2. aws emr **create**-**default**-**roles**

This command will create EMR\_EC2\_DefaultRole and EMR\_DefaultRole in your account.

1. **Launch your cluster** - Run the script below to launch your cluster. Be sure to include the appropriate file names within the <> in the code.
2. **[Optional] Specify your bootstrap file** - You should save an executable (bootstrap\_emr.sh file) in an accessible S3 location. You can specify this option as, for example, --bootstrap-actions Path=s3://mybucket/bootstrap\_emr.sh in the script below. A sample file is provided in the [**Github repo here**](https://github.com/udacity/nd027-c3-data-lakes-with-spark/tree/master/Setting_Spark_Cluster_In_AWS/exercises/starter/create_emr_cluster).
3. **# Add your cluster name, and EC2 private key file name**
4. aws emr create-cluster --name <YOUR\_CLUSTER\_NAME> --use-**default**-roles --release-label emr-5.28.0 --instance-count 3 --applications Name=Spark --ec2-attributes KeyName=<YOUR\_EC2\_KEY\_FILE\_NAME> --instance-type m5.xlarge --instance-count 3 --**auto**-terminate

Notice two things in the command above.

* + One, we have added the --auto-terminate option to terminate the cluster after completing all the steps because EMR clusters are costly. However, you can ignore this option, and [**terminate the cluster manually**](https://docs.aws.amazon.com/emr/latest/ManagementGuide/UsingEMR_TerminateJobFlow.html) after your job is done.
  + Two, we haven't specified the --bootstrap-actions option. This step is optional.

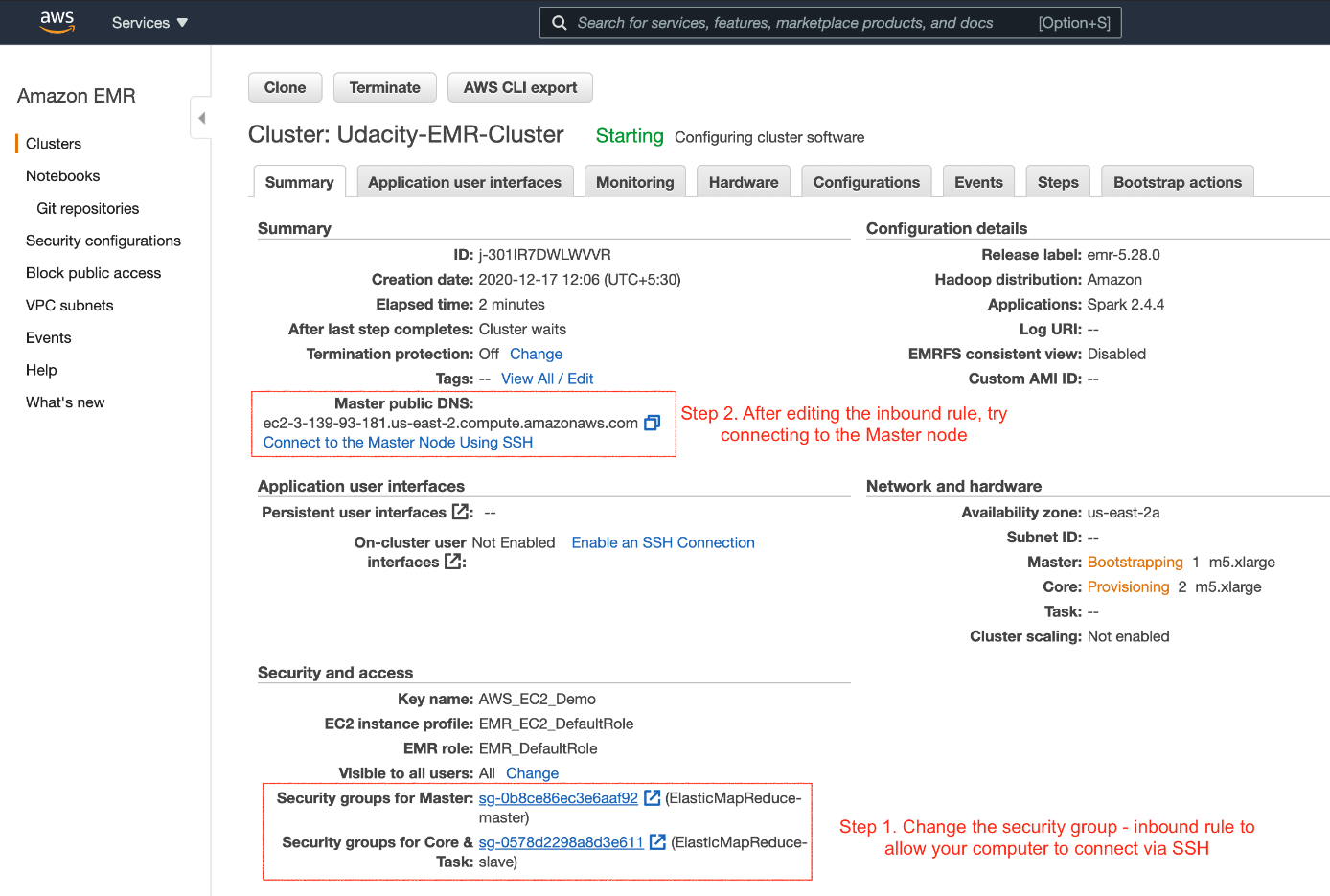
The expected output should look similar to this:

"ClusterId": "j-2PZ79NHXO7YYX",

"ClusterArn": "arn:aws:elasticmapreduce:us-east-2:027631528606:cluster/j-2PZ79NHXO7YYX"

1. You can either go to [**AWS EMR console**](https://console.aws.amazon.com/elasticmapreduce/home) from your web browser or run the command below to verify if the cluster is created successfully.
2. aws emr **describe**-cluster *--cluster-id <CLUSTER\_ID FROM ABOVE>*

A copy of the exercises are also available in the lesson git repo:[**Link to Github**](https://github.com/udacity/nd027-c3-data-lakes-with-spark/tree/master/Setting_Spark_Cluster_In_AWS/exercises/starter)



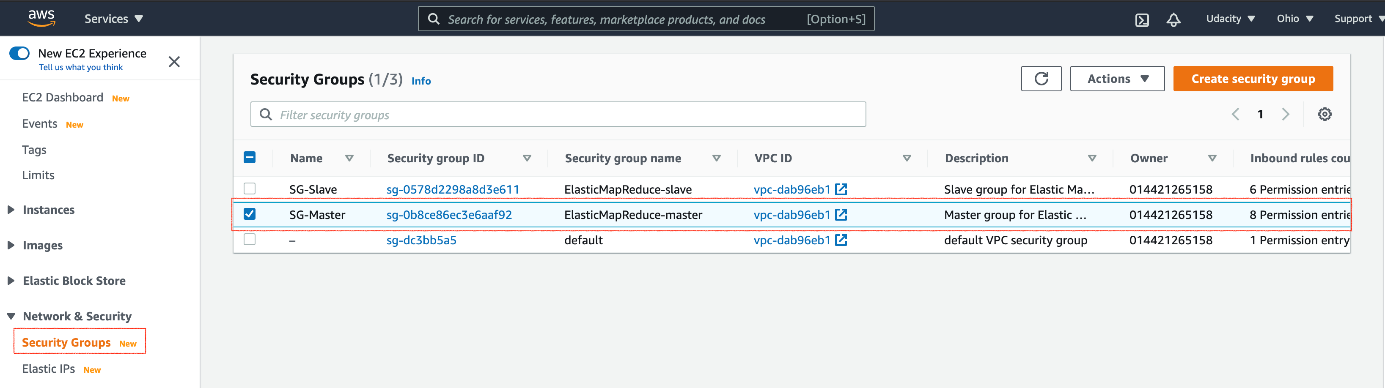
Summary of the newly created cluster. The next set of steps are also highlighted above.

1. **Troubleshoot** - Refer here if you get [**"EMR\_DefaultRole is invalid" or "EMR\_EC2\_DefaultRole is invalid"**](https://aws.amazon.com/premiumsupport/knowledge-center/emr-default-role-invalid/) error.

### 2.3. Change Security Groups

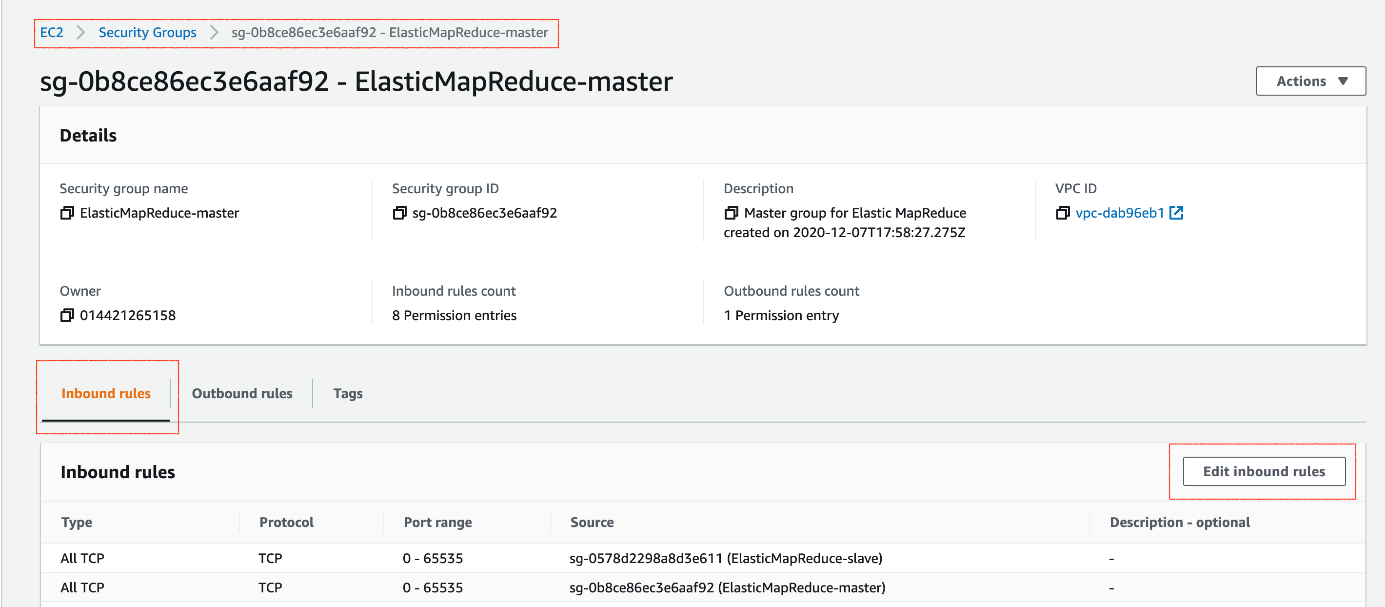
1. After successfully launching the EMR cluster, the master and core (slave) EC2 instances will launch automatically. Next, we will try to log in to the master EC2 instance on the EMR cluster using the SSH protocol (allows secure remote login). Therefore, you’ll need to enable the Security Groups setting of the master EC2 instance to accept incoming SSH protocol from your local computer.

The master and slave nodes are associated with a separate security group. You can view the security group ID either in the **EMR console** → **Clusters** or you can go to the **EC2 dashboard** → **Security Groups** service, as shown below.

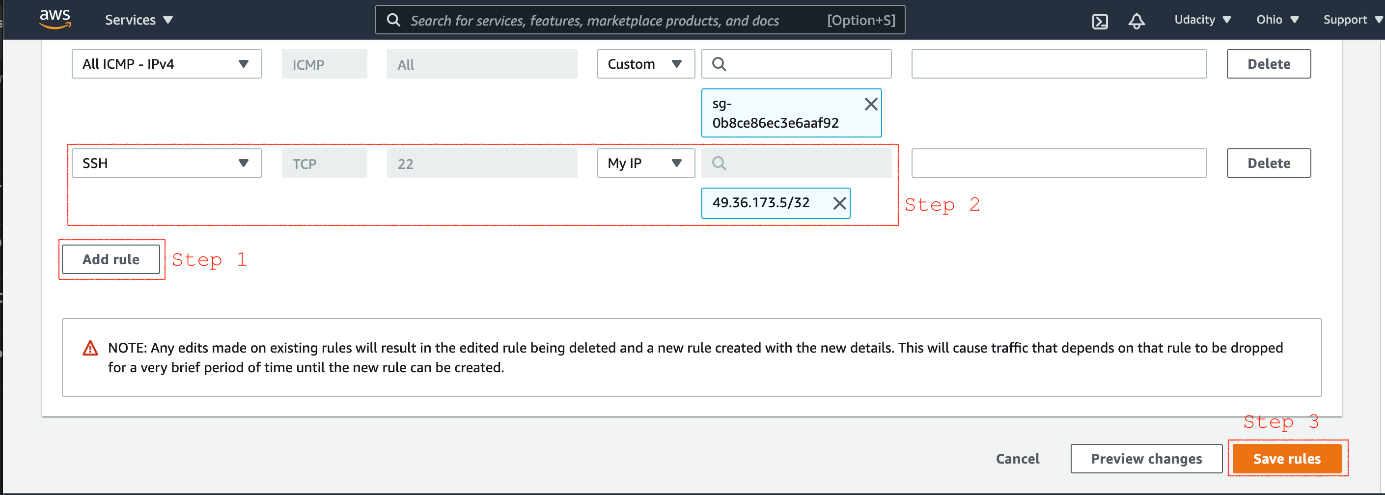


Select the security group associated with the master

1. Edit the security group to authorize inbound SSH traffic (port 22) from your local computer.



Edit the inbound rules of the master node

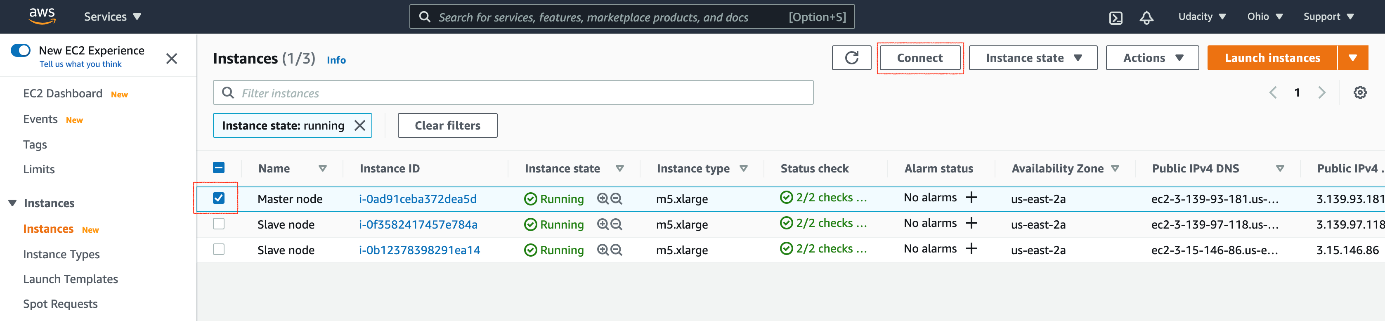


Add new inbound SSH traffic (port 22) from your local IP

1. **Reference** - [**Authorize inbound traffic**](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-ssh-prereqs.html)

### 2.4. Verify connection to the Master node

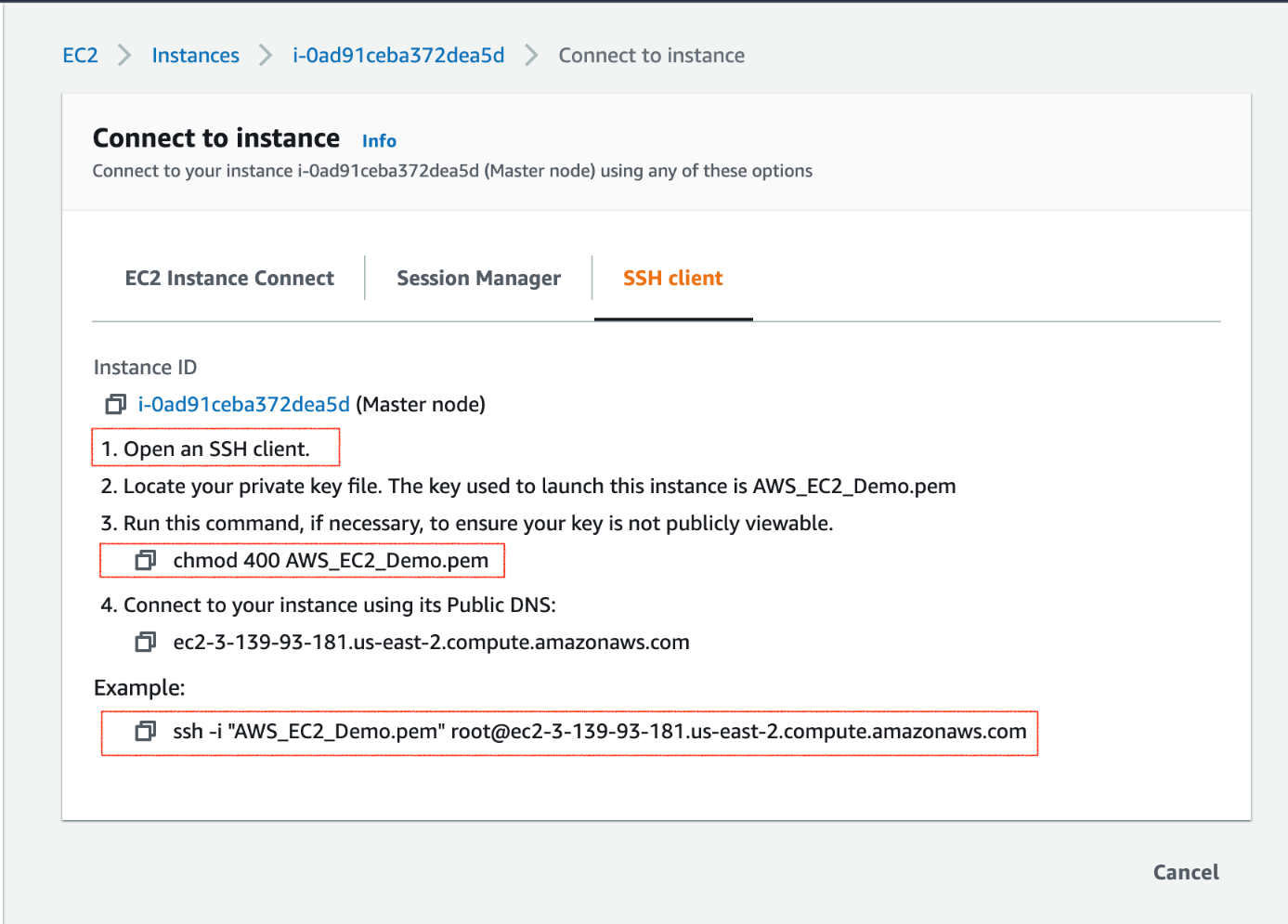
1. Go to the EC2 dashboard, and select the instance you want to connect using the SSH protocol.



Select the instance to connect

1. Connect using the SSH protocol. You can run the commands shown in the figure below in your terminal.

**Note** - In the snapshot below, the user name to log in is not **root**. Instead, you must use **hadoop**. For example, use ssh -i AWS\_EC2\_Demo.pem hadoop@ec2-3-139-93-181.us-east-2.compute.amazonaws.com



Steps to connect using SSH protocol. After a successful connection, you can exit your connection.

1. **Reference** - [**Connect to the Master Node Using SSH**](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-ssh.html).

### 2.5. View Spark UI hosted on the EMR Clusters

One last thing to do before using the Jupyter Notebook, or even browsing the Spark UI, is to set up a proxy in your browser. It is a two-step process.

**Step 1. Set Up an SSH Tunnel to the Master Node Using Dynamic Port Forwarding**

1. Enable the dynamic port forwarding using the command. This command does not returns a response.
2. ssh -i AWS\_EC2\_Demo.pem -N -D 8157 hadoop@ec2-3-139-93-181.us-east-2.compute.amazonaws.com

Replace the .pem file name and the master node public DNS for you. In the above example, the .pem is residing in the present working folder. If your .pem is placed in any different folder, you can provide the complete path.

In the command above, the -D option is used for specifying a local port (8157) to forward data to all remote ports on the master node's web server.

1. Now, you'd want to copy your .pem file (EC2 log in private key) to the master node. You can securely copy your .pem file from your local computer to the master node, using:
2. scp -i AWS\_EC2\_Demo.pem AWS\_EC2\_Demo.pem hadoop@ec2-3-139-93-181.us-east-2.compute.amazonaws.com:/home/hadoop/

You can use a similar command to copy any other script, if required.

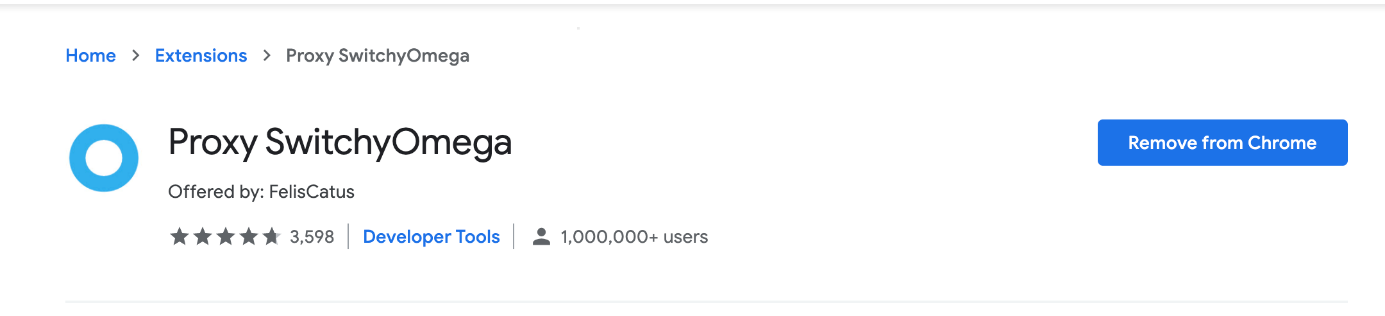
1. **Reference** - [**Part 1: Set Up an SSH Tunnel to the Master Node Using Dynamic Port Forwarding**](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-ssh-tunnel.html)

**Step 2. Configure Proxy Settings in your Local Computer**  
To do this, you'll need to install an extension in your browser. Here are the options:

* Chrome - SwitchyOmega or FoxyProxy
* Firefox - FoxyProxy

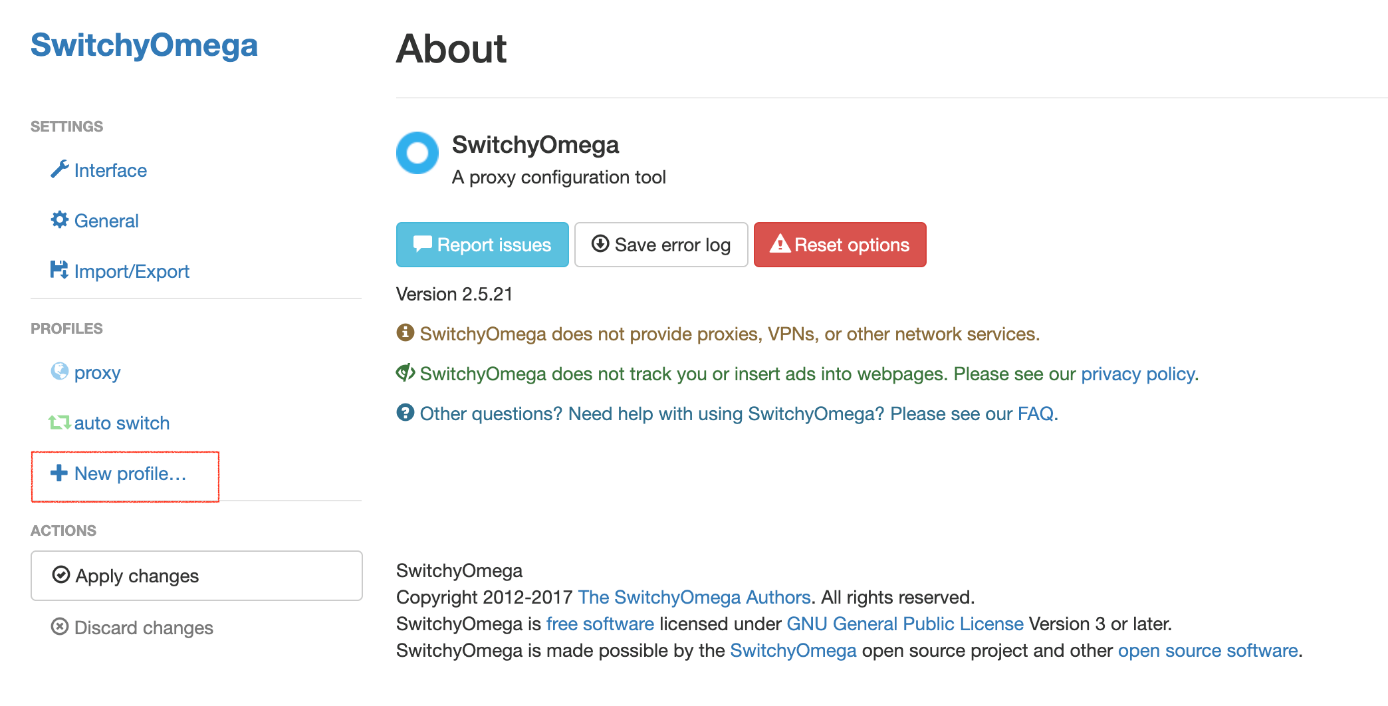
The snapshots below present the step for the Chrome browser. For other browsers, you can follow the reference link present at the end of the section.

1. Go to the [**https://chrome.google.com/webstore/category/extensions**](https://chrome.google.com/webstore/category/extensions), and add for Proxy SwitchyOmega extension to your Chrome browser.

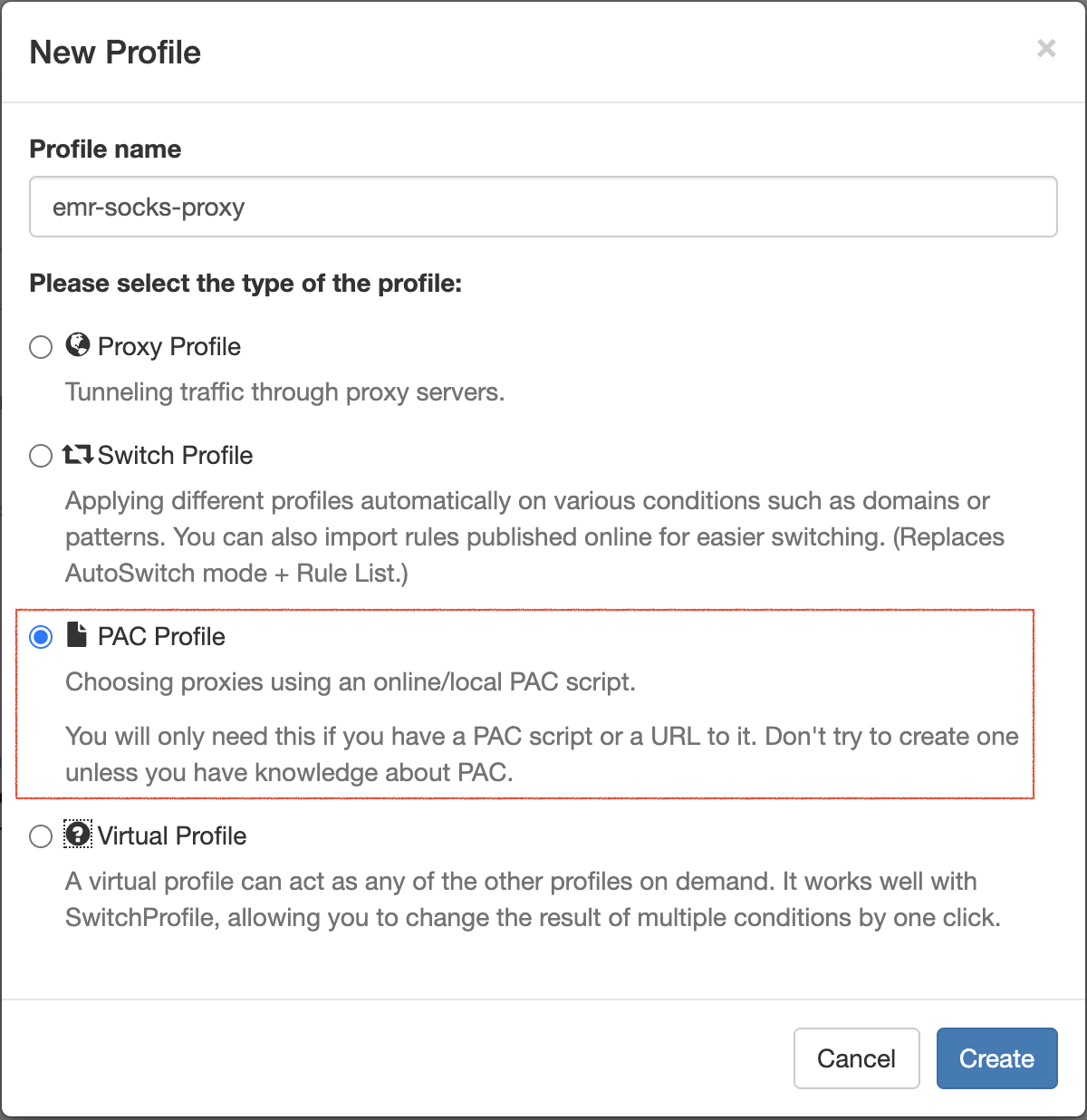


SwitchyOmega extension on Chrome

1. Create a new profile with name emr-socks-proxy and select PAC profile type.



Create a new profile in SwitchyOmega



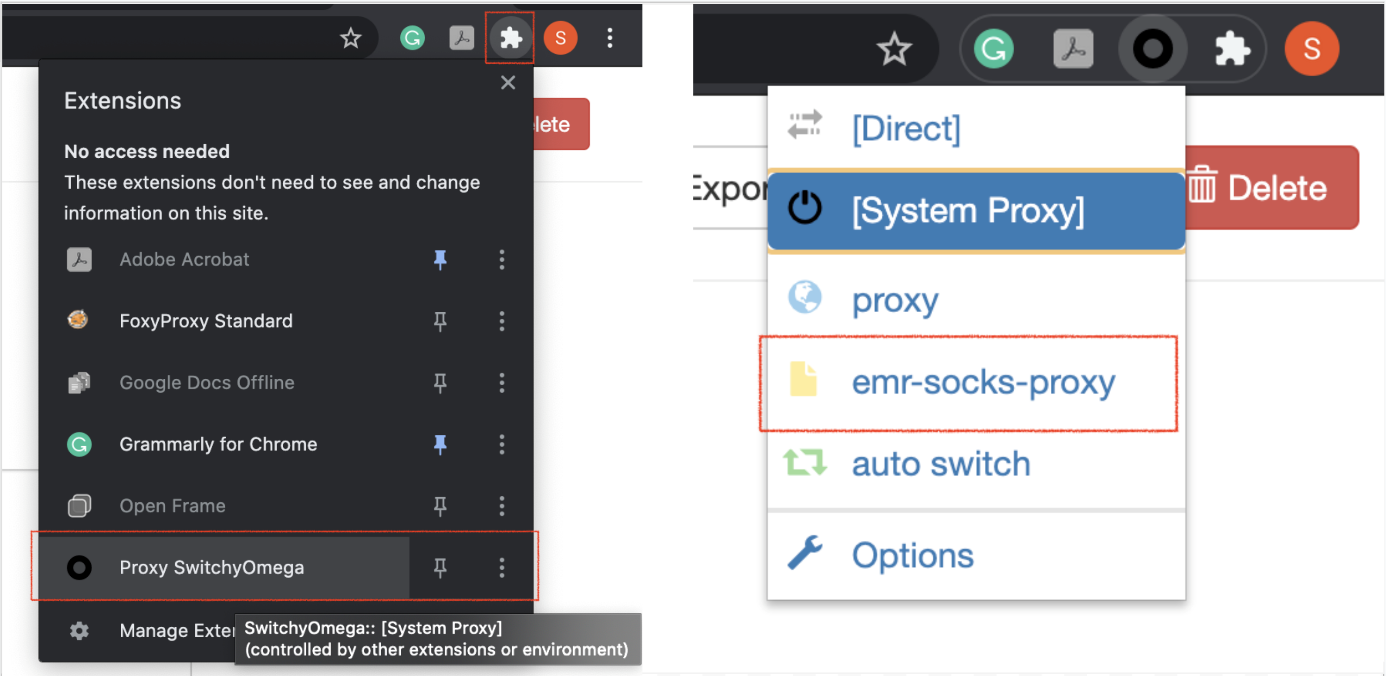
Enter the profile name and choose a profile type

1. Save the following profile script in your new profile:
2. **function** **FindProxyForURL**(url, host) {
3. **if** (shExpMatch(url, "\*ec2\*.amazonaws.com\*")) **return** 'SOCKS5 localhost:8157';
4. **if** (shExpMatch(url, "\*ec2\*.compute\*")) **return** 'SOCKS5 localhost:8157';
5. **if** (shExpMatch(url, "http://10.\*")) **return** 'SOCKS5 localhost:8157';
6. **if** (shExpMatch(url, "\*10\*.compute\*")) **return** 'SOCKS5 localhost:8157';
7. **if** (shExpMatch(url, "\*10\*.amazonaws.com\*")) **return** 'SOCKS5 localhost:8157';
8. **if** (shExpMatch(url, "\*.compute.internal\*")) **return** 'SOCKS5 localhost:8157';
9. **if** (shExpMatch(url, "\*ec2.internal\*")) **return** 'SOCKS5 localhost:8157';
10. **return** 'DIRECT';
11. }



Apply changes to the new profile

1. Enable the emr-socks-proxy profile.



Enable the new SwitchyOmega profile

1. Once, you have configured the proxy, you can access the Spark UI using the command (replace the master node public DNS for you):
2. http:*//ec2-3-139-93-181.us-east-2.compute.amazonaws.com:18080/*



Spark UI, accessed from the CLI (note the URL above). Though, you can access the same Spark UI by selecting the cluster summary from the **EMR console** → **Clusters**, and clicking on the **Persistent user interface** hyperlink.

1. **Reference** - [**Part 2: Configure Proxy Settings to View Websites Hosted on the Master Node**](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-proxy.html)

**Note** - Do not forget to [**Terminate**](https://docs.aws.amazon.com/emr/latest/ManagementGuide/UsingEMR_TerminateJobFlow.html) your EMR cluster after your exercise is finished.

6: Using notebooks on your cluster

Link: <https://www.youtube.com/watch?v=EcIYPkCkehY>

Notes:

# Jupyter / Zeppelin Notebook

There are a couple of options for which notebook to use. We can use a Jupyter Notebook, or use a Zeppelin notebook. If you are already familiar with Jupyter Notebooks, continue using them.

### Advantages of using Zeppelin Notebook

While the use of Jupyter Notebook is common across the industry, you can explore using Zeppelin notebooks. Zeppelin notebooks have been available since EMR 5.x versions, and they have direct access to Spark Context, such as a local spark-shell. For example, if you type sc, you’ll be able to get Spark Context within Zeppelin notebooks.

Zeppelin is very similar to Jupyter Notebook, but if you want to use other languages like Scala or SQL, on top of using Python, you can use Zeppelin instead.

7: Spark Scripts

Link: <https://www.youtube.com/watch?v=bfOocPv54EI>

Notes:

8: Submitting Spark Scripts

Link: <https://www.youtube.com/watch?v=ZcSfIqAgoUQ>

Notes:

### Submitting Spark Script Instructions

Here is the link to the [**GitHub repo**](https://github.com/udacity/nd027-c3-data-lakes-with-spark/tree/master/Setting_Spark_Cluster_In_AWS/exercises/starter) where a copy of the exercise instructions are located along with cities.csv file.

* Download the cities.csv dataset to your local machine.
* Upload a file into an S3 location using the AWS S3 console, or you can use the AWS CLI command, like aws s3 cp <your current file location>/<filename> s3://<bucket\_name>.
* Create an EMR instance.
* Copy the file to your EMR instance, preferably in your home directory of EMR instance.
* Execute the file using spark-submit <filename>.py.

##### A note about SSH

SSH is a specific protocol for secure remote login and file transfer.

The instructor is showing you one way to save your files. He is using SSH protocol to save the files in the EMR instance. When you see hadoop@ip-###- ###-####, this indicates that the instructor accessed the EMR instance using SSH protocol. However, once he terminates the EMR instance, everything he would saved on the EMR instance will be lost. This is because EMR instance is not kept active all the time since it is expensive.

In the Reflection Exercise you can experiment with an alternate good industry practice. Data engineers always save their initial, final, and intermediate data of the data pipeline in the S3 for future retrieval. It is best practice to move your files from your local machine to AWS S3, then use the program to read the data from AWS S3.

### Reflection exercise:

Use your proxy to view the Spark UI to understand how your code and workers are working, i.e. which are transformation vs action words (and if they are correctly showing up on Spark UI), and to get familiar with reading the Spark UI. This will give you a better understanding on how your Spark program runs.

**Reminder link** to [**Amazon documentation**](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-proxy.html) on FoxyProxy

<https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-proxy.html>

#### **Supporting Materials**

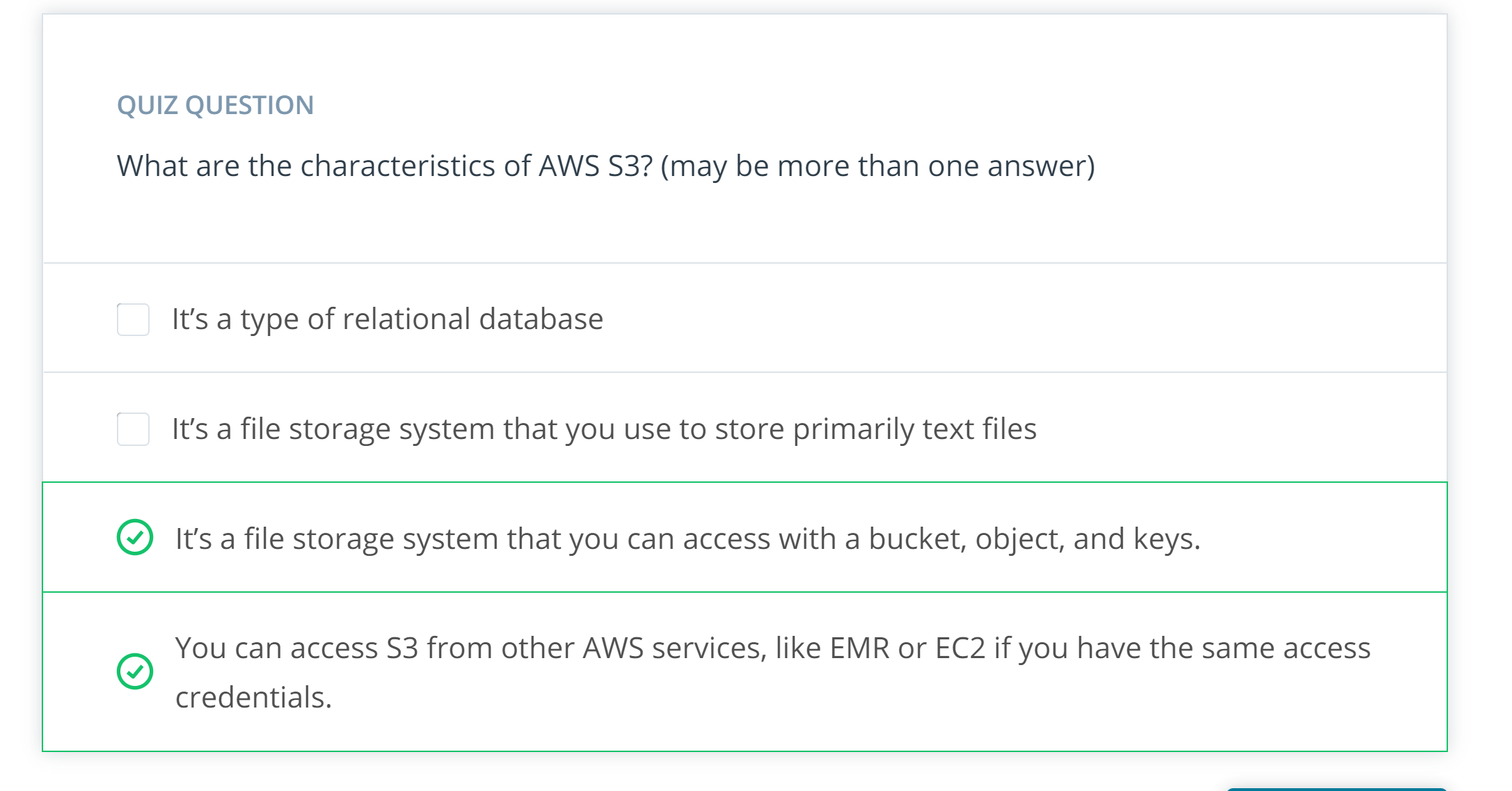
[**Cities**](https://video.udacity-data.com/topher/2020/May/5eabf02f_cities/cities.csv)

NEXT

9: Storing and Retrieving data on the cloud

Link: <https://www.youtube.com/watch?v=MrF2sHdpXJo>

Notes:



10: Reading and writing to Amazon S3

Link: <https://www.youtube.com/watch?v=j4kpT3DQ8i8>

Link2: <https://www.youtube.com/watch?v=yXfb4vwg7aM>

Notes:

### S3 Buckets

With the convenient AWS UI, we can easily mistake AWS S3 (Simple Storage Service) equivalent as Dropbox or even Google Drive. This is not the case for S3. S3 stores an object, and when you identify an object, you need to specify a bucket, and key to identify the object. For example,

df = spark.read.load(“s3://my\_bucket/path/to/file/file.csv”)

From this code, s3://my\_bucketis the bucket, and path/to/file/file.csv is the key for the object. Thankfully, if we’re using spark, and all the objects underneath the bucket have the same schema, you can do something like below.

df = spark.**read**.load(“s3:*//my\_bucket/”)*

This will generate a dataframe of all the objects underneath the my\_bucket with the same schema. Pretend some structure in s3 like below:

my\_bucket

|*---test.csv*

path/**to**/

|*--test2.csv*

**file**/

|*--test3.csv*

|*--file.csv*

If all the csv files underneath my\_bucket, which are test.csv, test2.csv, test3.csv, and file.csv have the same schema, the dataframe will be generated without error, but if there are conflicts in schema between files, then the dataframe will not be generated. As an engineer, you need to be careful on how you organize your data lake.

### [Link to Github Repo](https://github.com/udacity/nd027-c3-data-lakes-with-spark/tree/master/Setting_Spark_Cluster_In_AWS/demo_code) on Demo code referred to in video: [HERE](https://github.com/udacity/nd027-c3-data-lakes-with-spark/tree/master/Setting_Spark_Cluster_In_AWS/demo_code)

#### **Supporting Materials**

[**Reading And Writing To AmazonS3**](https://video.udacity-data.com/topher/2020/May/5ebba92c_reading-and-writing-to-amazons3/reading-and-writing-to-amazons3.zip)

11: Understanding difference between HDFS and Aws S3

Link: <https://www.youtube.com/watch?v=vsB_VLoiwyc>

Notes:

### Differences between HDFS and AWS S3

Since Spark does not have its own distributed storage system, it leverages using HDFS or AWS S3, or any other distributed storage. Primarily in this course, we will be using AWS S3, but let’s review the advantages of using HDFS over AWS S3.

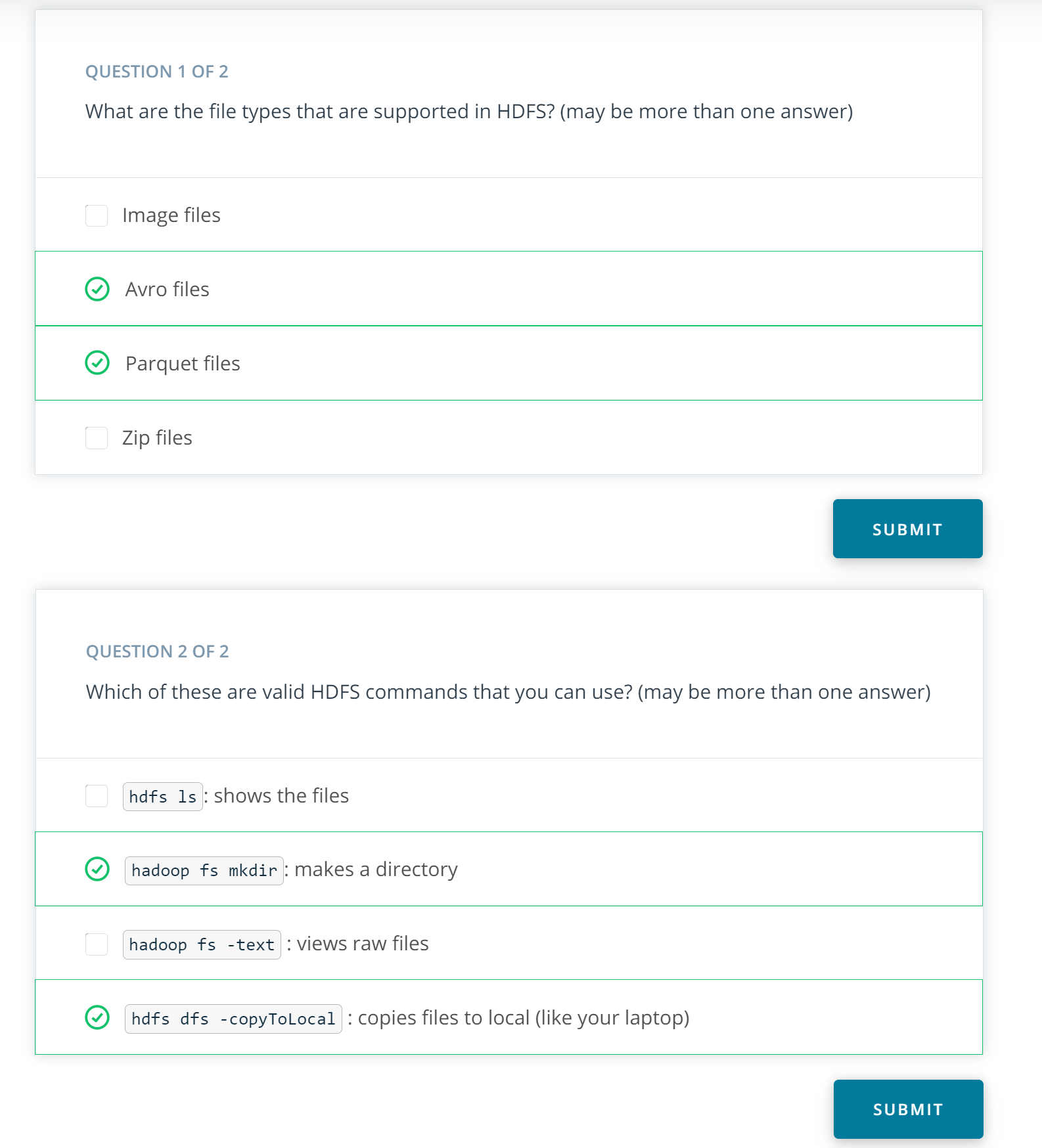
Although it would make the most sense to use AWS S3 while using other AWS services, it’s important to note the differences between AWS S3 and HDFS.

* **AWS S3** is an **object storage system** that stores the data using key value pairs, namely bucket and key, and **HDFS** is an **actual distributed file system** which guarantees fault tolerance. HDFS achieves fault tolerance by having duplicate factors, which means it will duplicate the same files at 3 different nodes across the cluster by default (it can be configured to different numbers of duplication).
* HDFS has usually been **installed in on-premise systems**, and traditionally have had engineers on-site to maintain and troubleshoot Hadoop Ecosystem, which **cost more than having data on cloud**. Due to the **flexibility of location** and **reduced cost of maintenance**, cloud solutions have been more popular. With extensive services you can use within AWS, S3 has been a more popular choice than HDFS.
* Since **AWS S3 is a binary object store**, it can **store all kinds of format**, even images and videos. HDFS will strictly require a certain file format - the popular choices are **avro** and **parquet**, which have relatively high compression rate and which makes it useful to store large dataset.

12: Reading and writing data to HDFS

Link: <https://www.youtube.com/watch?v=IVdbgtCLnmA>

Notes:



13: Recap local mode to cluster mode

Link: <https://www.youtube.com/watch?v=W434NZOxrhk>

Notes: