Setting up Spark Clusters with AWS

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

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.

Using Spark on AWS

When you want to rent a cluster of machines on AWS to run Spark, you have 2 choices:

1. Use AWS Elastic Compute (EC2) machines and install and configure

Spark and HDFS yourself
2. Use the AWS EMR service (previously called Elastic MapReduce), which is a scalable set of EC2 machines that are already configured to run Spark

EC2 vs EMR

	AWS EMR	**AWS EC2**
Distributed computing	Yes	Yes
Node categorization	Categorizes secondary nodes into core and task nodes as a result of which data can be lost in case a data node is removed.	Does not use node categorization
Can support HDFS?	Yes	Only if you configure HDFS on EC2 yourself using multi-step process.
What S3 protocol can be used?	Uses S3 protocol over AWS S3, which is faster than s3a protocol	ECS uses s3a
Comparison cost	Bit higher	Lower

Using Spark on AWS

When you want to rent a cluster of machines on AWS to run Spark, you have 2 choices:

1. Use AWS Elastic Compute (EC2) machines and install and configure Spark and HDFS yourself
2. Use the AWS EMR service (previously called Elastic MapReduce), which is a scalable set of EC2 machines that are already configured to run Spark

EC2 vs EMR

	AWS EMR	**AWS EC2**
Distributed computing	Yes	Yes

Node categorization	Categorizes secondary nodes into core and task nodes as a result of which data can be lost in case a data node is removed.	Does not use node categorization
Can support HDFS?	Yes	Only if you configure HDFS on EC2 yourself using multi-step process.
What S3 protocol can be used?	Uses S3 protocol over AWS S3, which is faster than s3a protocol	ECS uses s3a
Comparison cost	Bit higher	Lower

Circling Back on HDFS

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).

Why Would You Use an 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. AWS **EMR** is a service 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 EMR Clusters on AWS

The next part of the lesson will demonstrate how to create an EMR Cluster from the AWS Console, and from the AWS CLI. Review the AWS documentation to set up an EMR Cluster.

QUIZ QUESTION

What are some characteristics of the AWS EMR standalone mode? (may be more than one answer)

- [ ] It runs on your local machine
- [x] It is distributed
- [x] Spark is taking care of the resource management

QUIZ QUESTION

What are some characteristics of the AWS EMR standalone mode? (may be more than one answer)

- [ ] It runs on your local machine
- [x] It is distributed
- [x] Spark is taking care of the resource management

Launch the AWS Web Console from your Udacity Classroom

You are given a *federated user account*, a temporary AWS user account with limited permissions, that you can use in this program.
Before you begin, be sure to **log out** of any AWS instances you may already have running. Then click on the **“LAUNCH CLOUD GATEWAY”** link in the left navigation pane to access a customized instance with the right permissions specific to this program. It will open up a pop-up, and clicking on the **"Open Cloud Console"** button in the pop-up will open the AWS console in a new browser tab. This may take a few moments to load the first time. See a brief video below for a walkthrough.

Ensure that you do not have a pop-up blocker installed; it may prevent the new tab from launching. Please be sure to allow pop-ups from Udacity.

Important Points to Remember

1. Session limit

Note that there is a certain **session time limit**. If reached, you will automatically be timed out. As long as you have not used your entire allocated budget, your work will be saved. You can re-launch using the same "Launch Cloud Gateway" button in the left navigation menu to return to your session.

2. Default AWS region

The default AWS region for you will be US East (N. Virginia) (us-east-1). In case you need more than one region for your work you can use us-east-2, us-west-1 or us-west-2. Note: You would not have permissions to access regions other than us-east-1, us-east-2, us-west-1, and us-west-2.

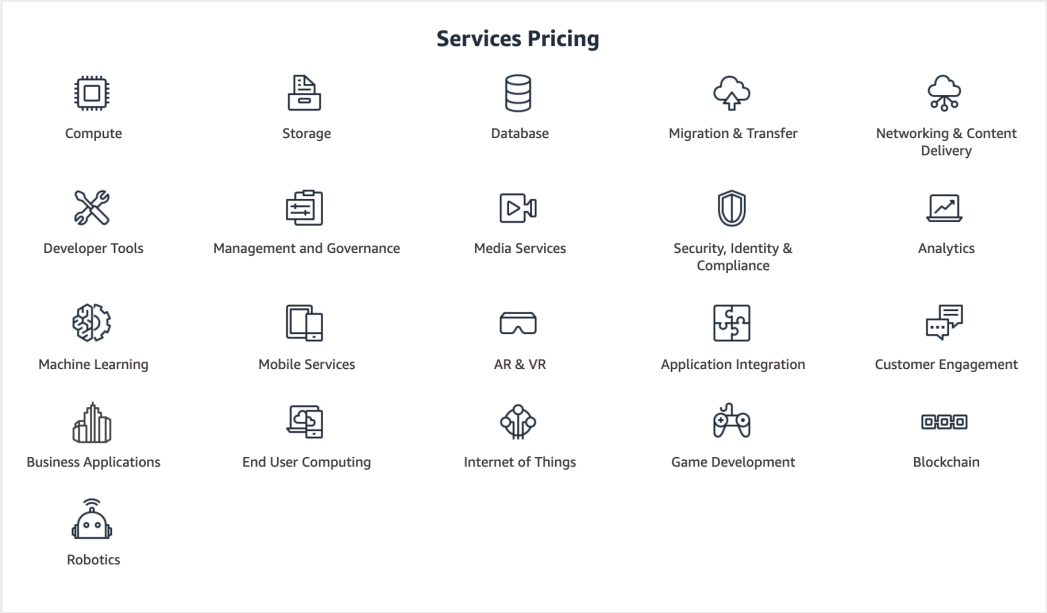
3. The budget allocated for you

All AWS services are a pay-as-you-go service. Udacity has set a budget for each student to complete their course work. Please understand that these credits are limited and available for you to use judiciously. **The budget for this entire course is $25.** Although, we find about $10 sufficient for most to complete this course.

4. Shut down your resources | No extra credits

We recommend you **shut down/delete every AWS resource** (e.g., EC2, Sagemaker, Database, EMR, CloudFormation) immediately after the usage or if you are stepping away for a few hours. Otherwise, you will run out of your allocated budget. **Udacity will not provide additional credits.** In case you exhaust your credits:

- **You will lose your progress on the AWS console.**
- **You will have to use your personal AWS account to finish the remaining ND.** Even if you are in the middle of the project/exercise and need to step away, you must shut down your resources. You can re-instantiate them later. To better understand pricing, see the AWS Pricing for all available services. > For reference, any service available to you @$0.1/hour or higher should be monitored closely and shut down immediately after use or if you are stepping away. >

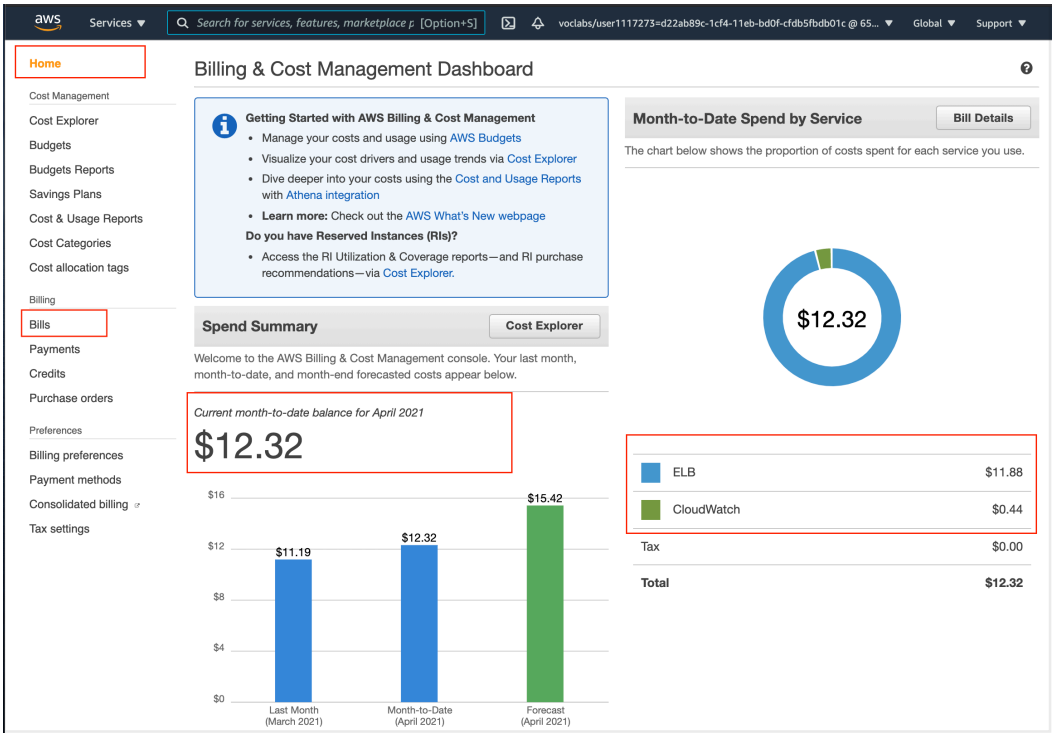


Check the pricing at https://aws.amazon.com/pricing/

5. Tracking your usage

You need to make sure that you have an adequate budget available to complete your project/task. **If you hit your budget, your session will time out and your work will be lost and unrecoverable.** Options for you to stay vigilant are:

- Track your usage on the AWS web console. Go to AWS Billing Dashboard, and view the monthly spending. It will list you the services constituting the spend.
- Submit a ticket to Student Support Services to know your current balance.



AWS Billing dashboard

Note *-*
As you are given a temporary AWS user account with ***limited*** *permissions, you might not be able to avail* ***all*** *AWS services. We have allowed the necessary ones only. If you see a few warning messages related to insufficient permissions, you can ignore them and proceed with your practice.*

Installing and Configuring the AWS CLI

You'll need to use the AWS CLI if you want to create or interact with an EMR Cluster using commands. Take these important steps to install and configure it.

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, you need to follow three steps:

1. Install AWS CLI
2. Create an IAM user with Administrator permissions
3. Configure the AWS CLI

Step 1. Install AWS CLI v2

Refer to the official AWS instructions to install/update AWS CLI (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.

Mac/Linux/Windows: Verify the successful installation of AWS CLI 2

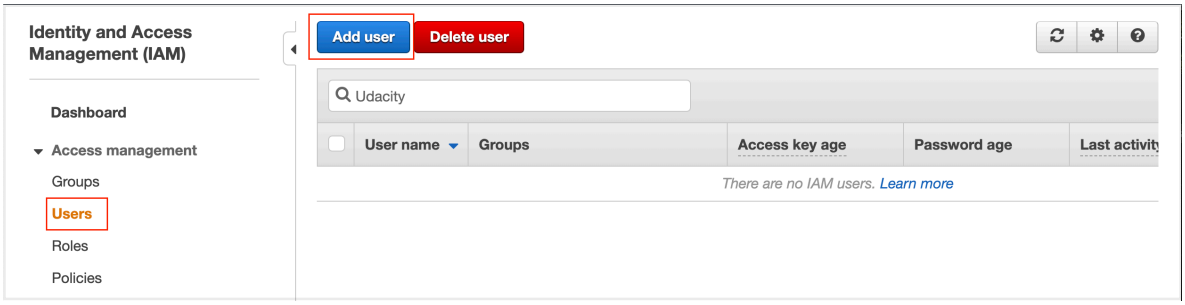
If You Are Using Your Own AWS Account:
Follow these instructions in Step 2 to create an IAM user if you are using your own personal or business AWS account. If you are using the Udacity-provided AWS Gateway and AWS account, you can skip this step.

Step 2. Create an IAM user (if you are using your own AWS account)
In this step, you will create an IAM user with Administrator permissions who is allowed to perform *any* action in your AWS account, only through CLI. After creating such an IAM user, we will use its **Access key** (long-term credentials) to configure the AWS CLI locally.
Let's create an AWS IAM user, and copy its Access key.

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

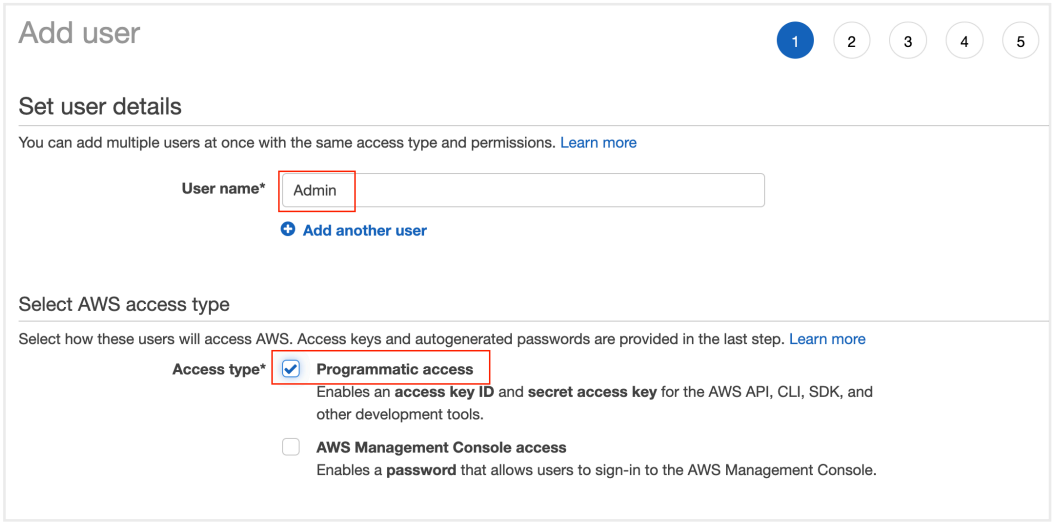
The Access key is a combination of an **Access Key ID** and a **Secret Access Key.** Let's see the steps to create an IAM user, and generate its Access key.

- Navigate to the IAM Dashboard, and create an IAM user.



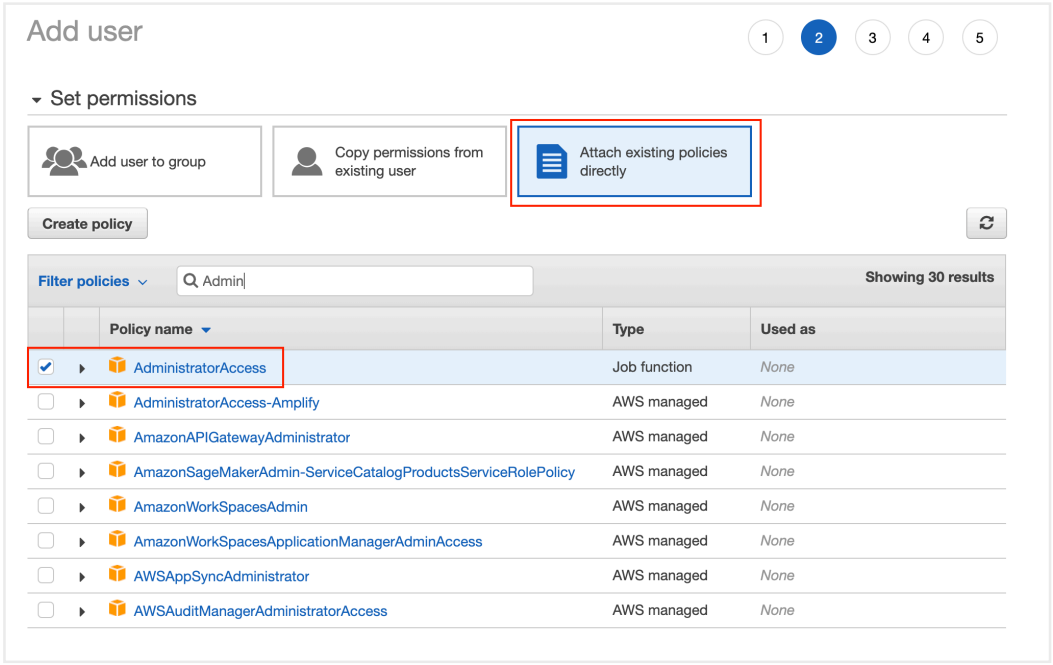
Add a new IAM User

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



Set the username, and type (mode) of access

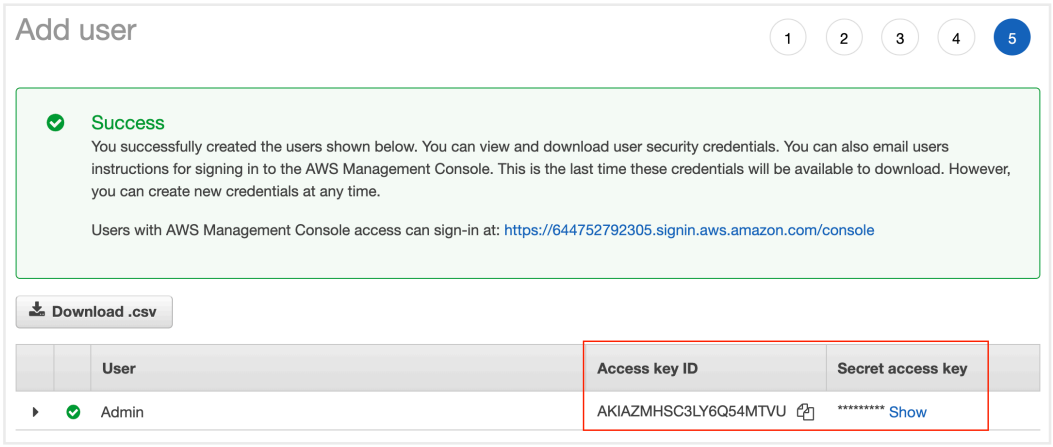
- Set the permissions to the new user by attaching the AWS Managed **AdministratorAccess** policy from the list of existing policies.



Attach the adminstratorAccess 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 OR download the .csv file containing the Access key

Step 3. Configure the AWS CLI

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

1. **Access key** - It is a combination of an *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*. If you are using the Udacity-provided AWS Gateway and account, these credentials are provided in the popup that appears when you click **"Launch AWS Gateway"** in the course menu to the left.
2. **Default AWS Region** - It specifies the AWS Region where you want to send your requests by default.
3. **Default output format** - It specifies how the results are formatted. It can either be a json, yaml, text, or a table.
4. **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.
5. **Session Token -** If you are using the Udacity-provided AWS Gateway and Account, you will also need to add the session token from the AWS Gateway credential popup, as well as the Access Key and SecretKey from the popup as described here.

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:

```
(base) xyz$ aws configure list
      Name                    Value             Type    Location
      ----                    -----             ----    --------
   profile                <not set>             None    None
access_key                <not set>             None    None
secret_key                <not set>             None    None
    region                us-east-2      config-file    /Users/xyz/.aws/config
(base) xyz$ aws configure --profile default
AWS Access Key ID [None]: [redacted]
AWS Secret Access Key [None]: [redacted]
Default region name [us-east-2]: us-east-2
Default output format [json]: json
(base) xyz$
(base) xyz$
```

Mac/Linux: List your present configuration, and then configure your default aws profile

- Set the default profile credentials

```
# Navigate to the home directory
cd ~
# If you do not use the profile-name, a default profile will be created for you.
aws configure --profile <profile-name>
# View the current configuration
aws configure list --profile <profile-name>
# View all existing profile names
aws configure list-profiles
# In case, you want to change the region in a given profile
# aws configure set <parameter> <value>  --profile <profile-name>
aws configure set region us-east-1  --profile <profile-name>
```

Moving forward, you can use --profile <profile-name> option with any AWS command. This will resolve the conflict if you have multiple profiles set up locally.

- Let the system know that your sensitive information is residing in the .aws folder

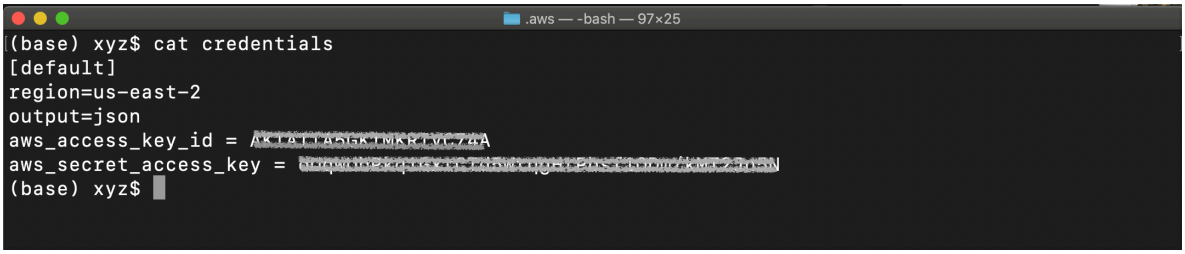
```
export AWS_CONFIG_FILE=~/.aws/config
export AWS_SHARED_CREDENTIALS_FILE=~/.aws/credentials
```

```
(base) xyz$ aws configure list
      Name                    Value             Type    Location
      ----                    -----             ----    --------
   profile                <not set>             None    None
access_key     ****************C74A shared-credentials-file
secret_key     ****************Jn5N shared-credentials-file
    region                us-east-2      config-file    /Users/xyz/.aws/config
(base) xyz$
```

Mac/Linux: A successful configuration

Setting the Session Token
If you are using the Udacity-Provided AWS Credentials from the AWS Gateway button in the classroom, you'll have to add the session token to the credential file. Refer to the AWS documentation on how to set this piece of your credentials

- After a successful credential set-up, your "credentials" file will look like this one below. If you are using the Udacity-provided AWS account, you should have a session token name / value pair in your configuration as well.

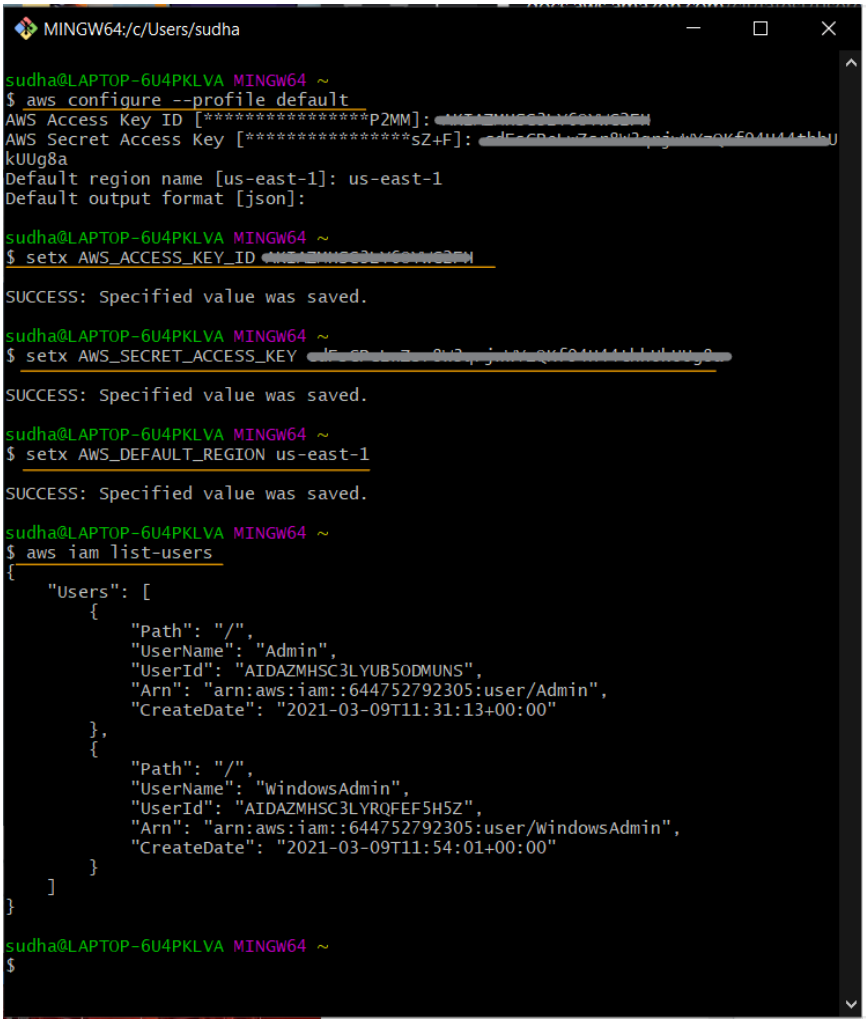


Mac/Linux: View the credentials file using cat ~/.aws/credentials command

- **Windows users with GitBash only** You will have to set the environment variables. Run the following commands in your GitBash terminal:

```
setx AWS_ACCESS_KEY_ID AKIAIOSFODNN7EXAMPLE
setx AWS_SECRET_ACCESS_KEY wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
setx AWS_DEFAULT_REGION us-west-2
(For Udacity AWS Gateway Users)
setx AWS_SESSION_TOKEN FwoGZXIvYXdzEAAaDIkHKlx8....
```

Replace the access key ID and secret, as applicable to you. Windows users using WSL do not need this step, they will follow all steps as if they are Linux users.



Windows: Successful configuration using the GitBash terminal

Step 4. Run your first AWS CLI command

- Check the successful configuration of the AWS CLI, by running either of the following AWS command:

```
# If you've just one profile set locally
aws iam list-users
# If you've multiple profiles set locally
aws iam list-users --profile <profile-name>
```

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"
    }
]
}
```


Troubleshoot

If you are facing issues while following the commands above, refer to the detailed instructions here -

1. Configuration basics
2. Configuration and credential file settings
3. Environment variables to configure the AWS CLI
4. Using the Session Token

Updating the specific variable in the configuration

In the future, you can set a single value, by using the command, such as:

```
# Syntax
# aws configure set <varname> <value> [--profile profile-name]
 aws configure set default.region us-east-2
```

It will update only the region variable in the existing default profile.

Creating an AWS EMR Cluster

There are 2 ways to create an EMR cluster

1. Through the AWS Console, as shown in the video below
2. Using the AWS CLI as described on the next page

Exercise: Create an EMR Cluster from the AWS Console

Follow the steps in the video to create an EMR Cluster with the recommended configuration.

Terminate Your Cluster to Avoid Excess Billing Charges

Note

- *Do not forget to **Terminate** your EMR cluster after your exercise is finished.*

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

Creating an EMR Cluster with the AWS CLI

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

aws emr create-cluster command

You can use theaws emr create-cluster command to create an EMR cluster. The advantage of this is you can reuse the command below multiple times:

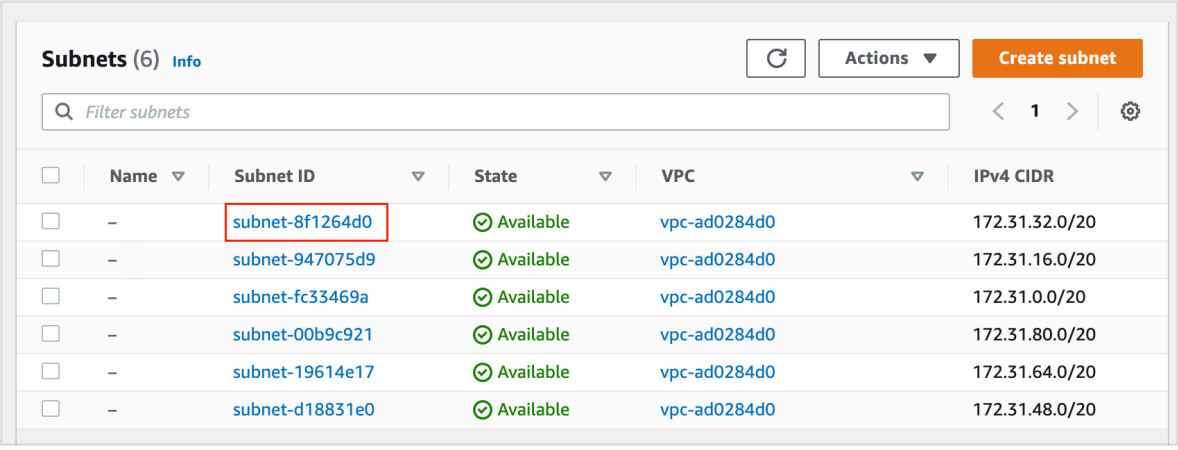
```
aws emr create-cluster --name <cluster_name> \
 --use-default-roles --release-label emr-5.28.0  \
--instance-count 3 --applications Name=Spark Name=Zeppelin  \
--bootstrap-actions Path="s3://bootstrap.sh" \
--ec2-attributes KeyName=<Key-pair-file-name>, SubnetId=<subnet-Id> \
--instance-type m5.xlarge --log-uri s3:///emrlogs/
```

Options: Let’s break down the command above and go over each option.

- --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: The KeyName field specifies your key-pair file name, for example, if it is *MyKey.pem*, just specify MyKey for this field. There is one more field that you should specify, SubnetId.
  The aws documentation says that the cluster must be launched within an EC2-VPC. Therefore, you need to provide the VPC subnet Id in which to create the cluster. If you do not specify this value, the cluster is launched in the normal AWS cloud, outside of any VPC. Go to the VPC service in the web console to copy any of the subnet IDs within the default VPC. If you do not see a default VPC in your account, use a simple command to create a default VPC:

```
aws ec2 create-default-vpc --profile <profile-name>
```

- See the snapshot below to copy the subnet Id.
- --instance-type: Specify the type of instances you want to use. Detailed list can be accessed here, 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.



Use the subnet ID in --ec2-attributes KeyName=<Key-pair-file-name>, SubnetId=<subnet-Id> option

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.

Exercise: Create an EMR cluster using AWS CLI
Follow the instructions to create an EMR cluster with the AWS CLI

Prerequisite

1. **AWS CLI** - Install AWS CLI on your local computer from the instructions on the previous page.
2. **Set up Access credentials using AWS IAM** - Follow the instructions on the previous page if you need to configure IAM credentials in the AWS CLI
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. 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.

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.

```
aws emr create-default-roles --profile <profile-name>
```

This command will create *EMR_EC2_DefaultRole* and *EMR_DefaultRole* in your account. If the role already exists then the command returns nothing.

Launch your cluster - Run the script below to launch your cluster. Be sure to replace the appropriate names within the <> in the command below.

```
# Provide cluster name, EC2 private key file name, and profile name
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=<Key-pair-file-name>, SubnetId=<subnet-Id> \
--instance-type m5.xlarge \
--auto-terminate \
--profile <profile-name>
```

Notice two things in the command above.

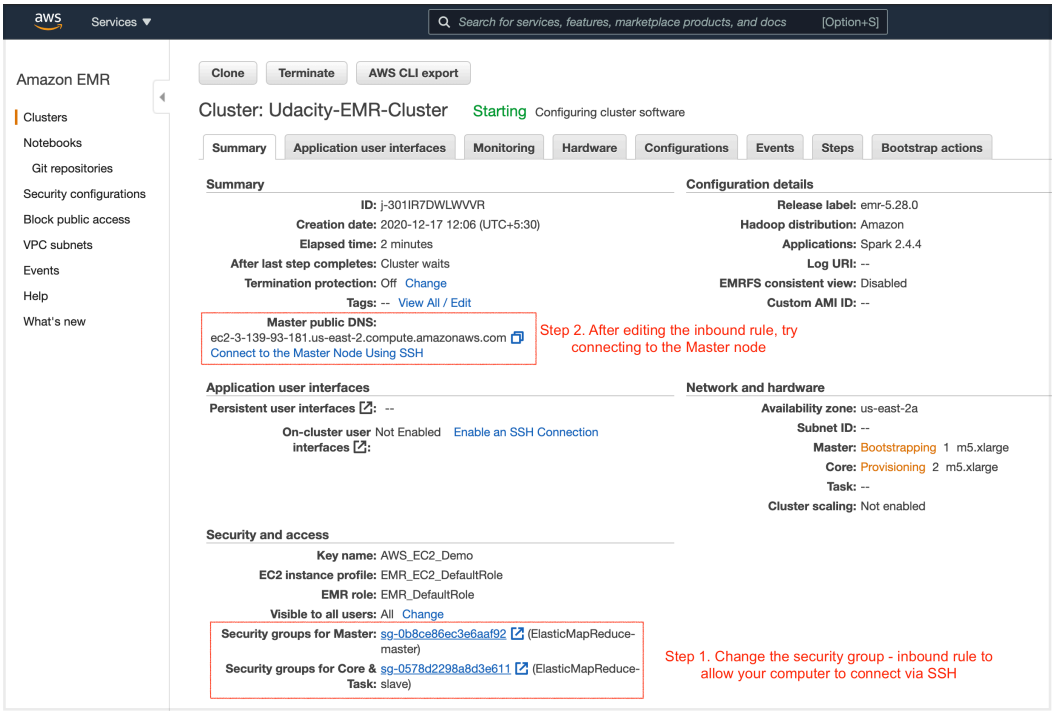
- One, we have added the --auto-terminate option to terminate the cluster after completing all the steps because EMR clusters are costly. However, you can ignore this option, and terminate the cluster manually after your job is done.
- Two, we haven't specified the --bootstrap-actions option. This step is optional.
- [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 command below. A sample file is provided in the Github repo here. The expected output should look similar to this:

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

You can either go to AWS EMR console from your web browser or run the command below to verify if the cluster is created successfully.

```
# Provide cluster ID and the profile name
aws emr describe-cluster --cluster-id <CLUSTER_ID FROM ABOVE> --profile <profile-name>
```

A copy of the exercises are also available in the lesson git repo: **Link to Github**



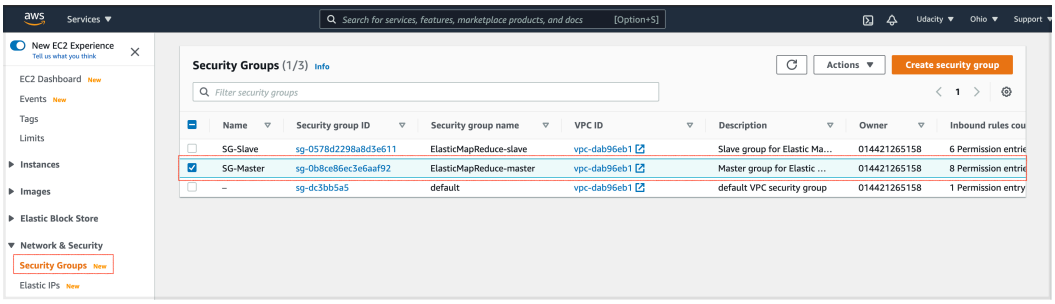
Summary of the newly created cluster. The next set of steps are also highlighted above.

5. **Troubleshoot** - Refer here if you get "EMR_DefaultRole is invalid" or "EMR_EC2_DefaultRole is invalid" error.

2.3. Change Security Groups

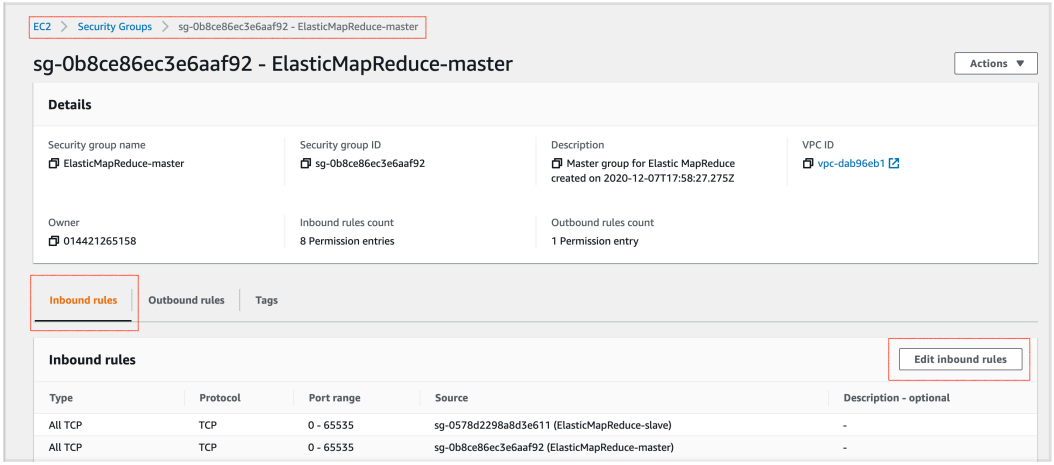
1. After successfully launching the EMR cluster, the master and core (slave) EC2 instances will launch automatically. Next, we will try to log in to the master EC2 instance on the EMR cluster using the SSH protocol (allows secure remote login). Therefore, you'll need to enable the *Security Groups* setting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

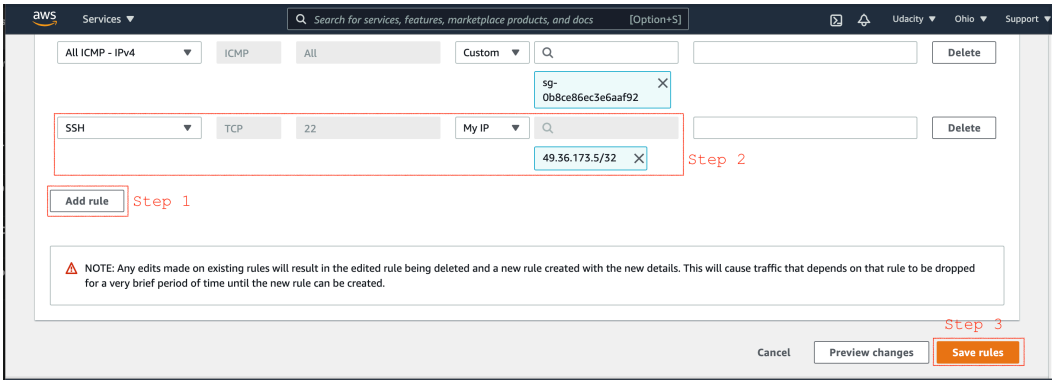


Select the security group associated with the master

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



Edit the inbound rules of the master node

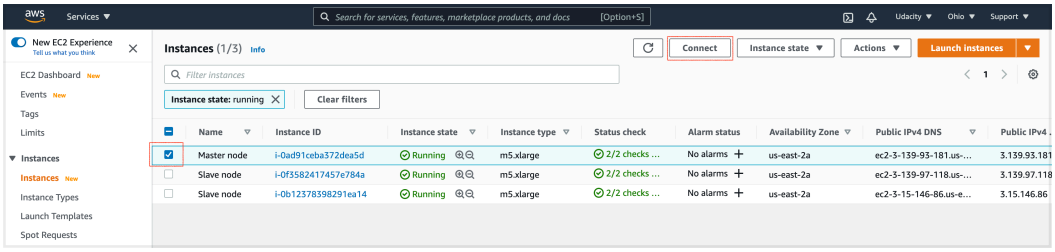


Add new inbound SSH traffic (port 22) from your local IP

3. **Reference** - Authorize inbound traffic

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.

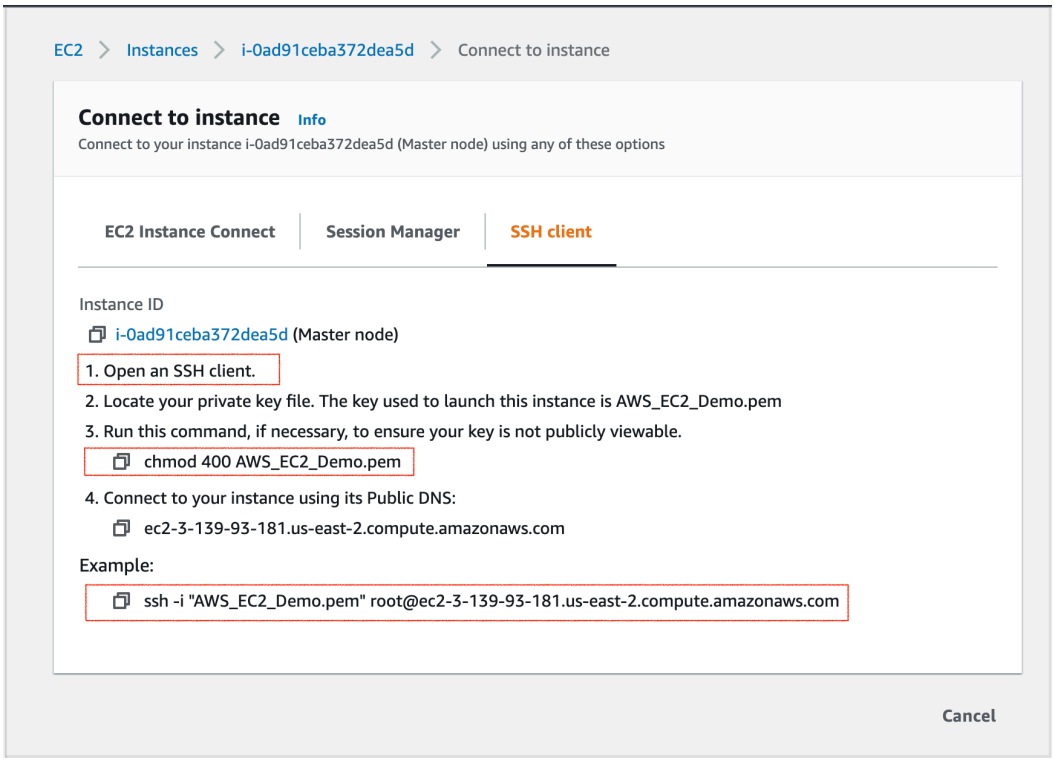


Select the instance to connect

2. 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.

3. **Reference** - Connect to the Master Node Using SSH.

2.5 View Spark UI hosted on the EMR Clusters

There are 2 ways to view the Spark UI hosted on the EMR Clusters

Option 1: Use the AWS Console in the Browser

You can access the Spark UI by selecting the cluster summary from the **EMR console** → **Clusters**, and clicking on the **Persistent user interface** hyperlink tab. Read more about accessing the Spark UI this way in the AWS Documentation.

Option 2: Set up a Web Proxy

Setting up a proxy in your browser to view the SparkUI 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.

```
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:

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.

2. **Reference** - Part 1: Set Up an SSH Tunnel to the Master Node Using Dynamic Port Forwarding

Step 2. Configure Proxy Settings in your Local Computer

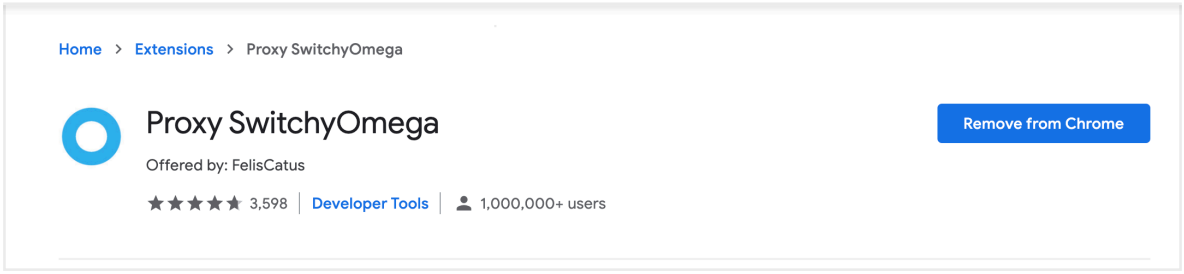
To do this, you'll need to install an extension in your browser. Here are the

options:

- Chrome - SwitchyOmega or FoxyProxy
- Firefox - FoxyProxy
-

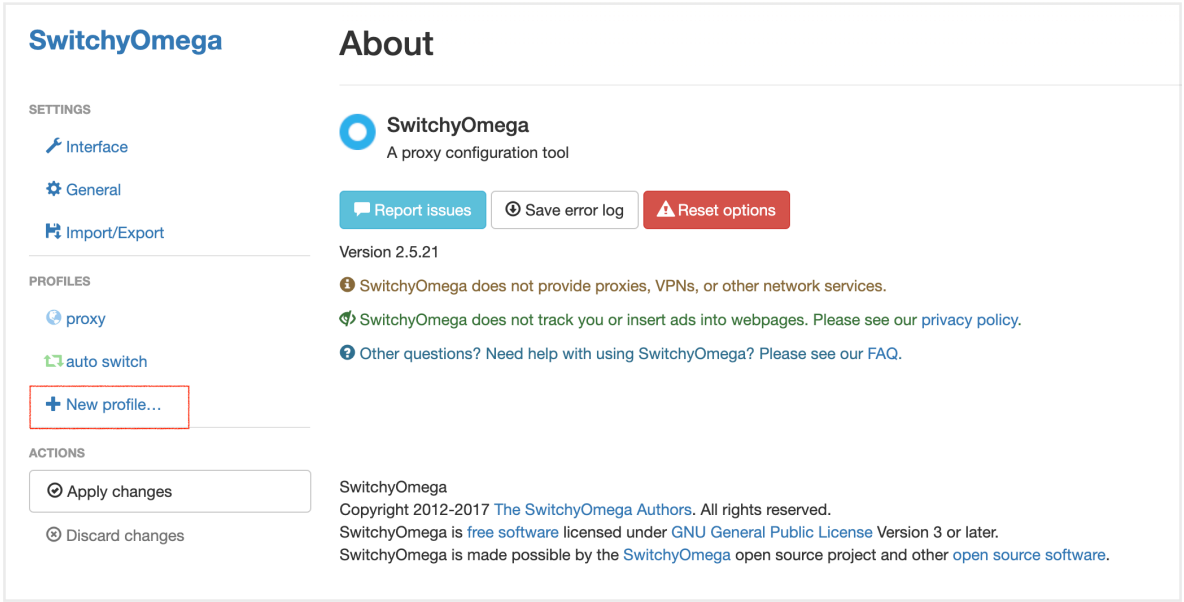
The snapshots below present the step for the Chrome browser. For other browsers, you can follow the reference link present at the end of the section.

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

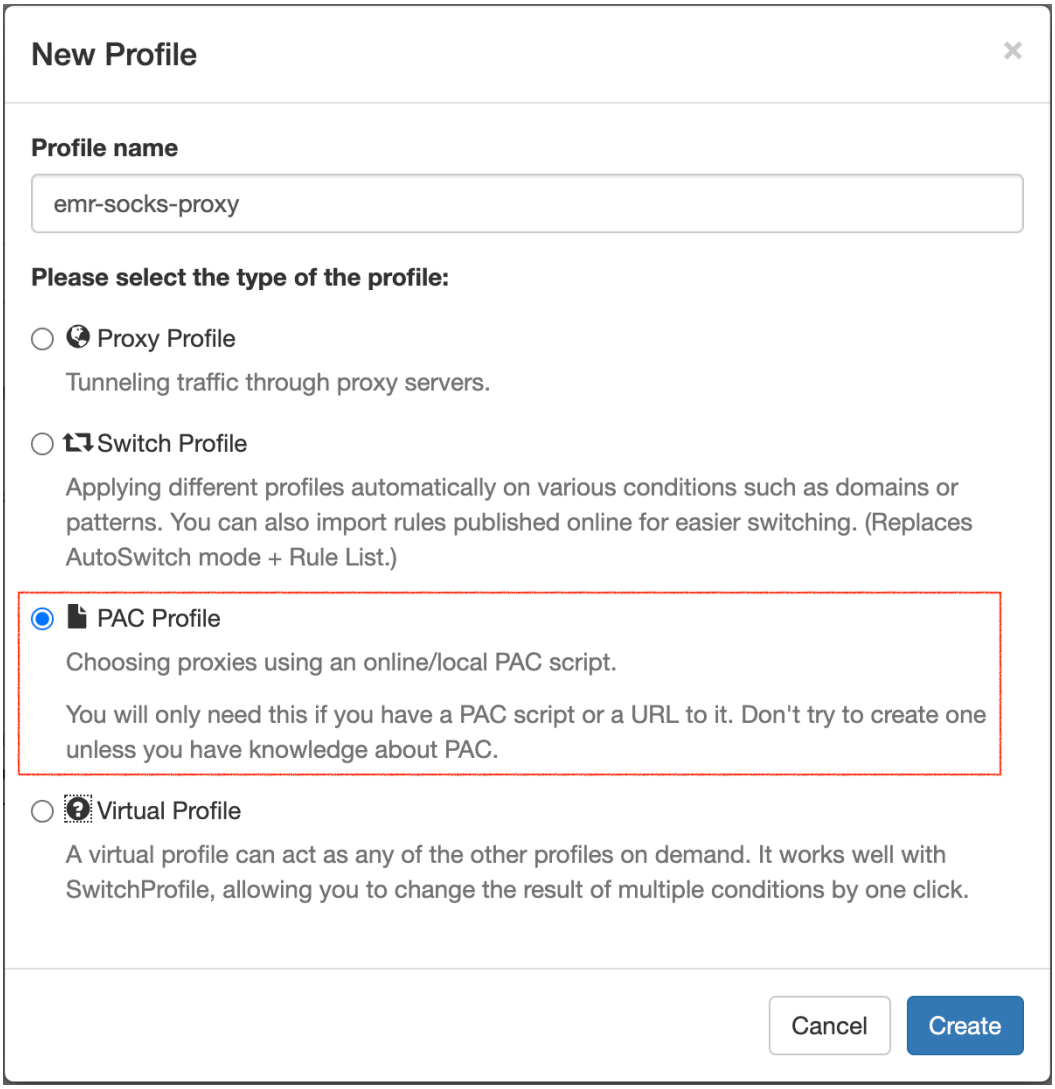


SwitchyOmega extension on chrome

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



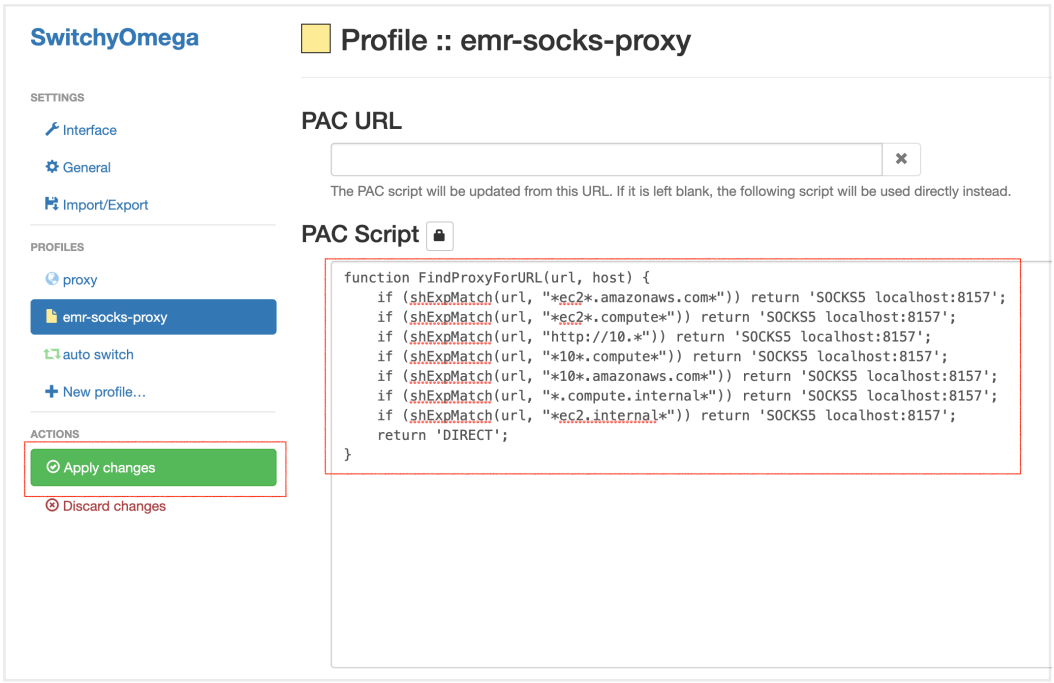
Create a new profile in SwitchyOmega



Enter the profile name and choose a profile type

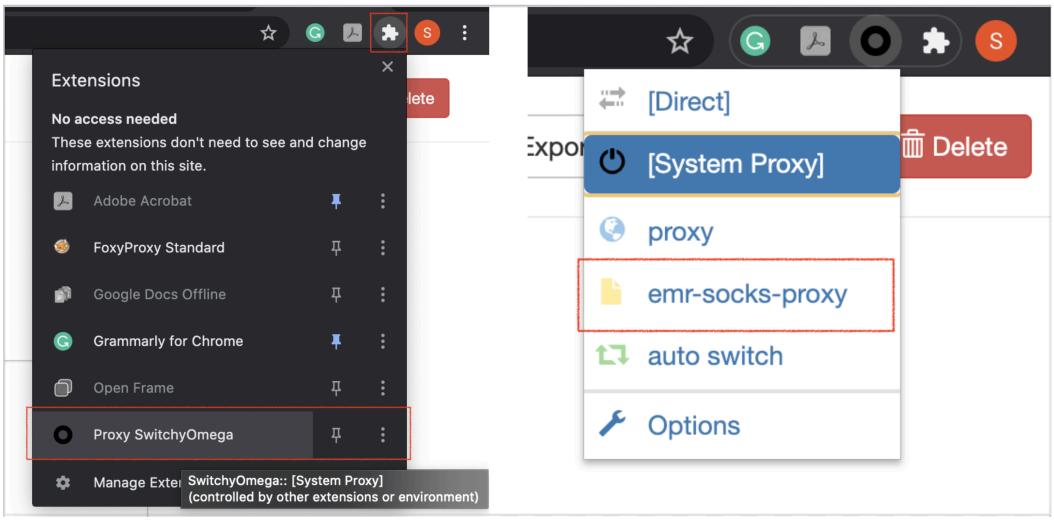
3. Save the following profile script in your new profile

```
function FindProxyForURL(url, host) {
 if (shExpMatch(url, "*ec2*.amazonaws.com*")) return 'SOCKS5 localhost:8157';
 if (shExpMatch(url, "*ec2*.compute*")) return 'SOCKS5 localhost:8157';
 if (shExpMatch(url, "http://10.*")) return 'SOCKS5 localhost:8157';
 if (shExpMatch(url, "*10*.compute*")) return 'SOCKS5 localhost:8157';
 if (shExpMatch(url, "*10*.amazonaws.com*")) return 'SOCKS5 localhost:8157';
 if (shExpMatch(url, "*.compute.internal*")) return 'SOCKS5 localhost:8157';
 if (shExpMatch(url, "*ec2.internal*")) return 'SOCKS5 localhost:8157';
 return 'DIRECT';
}
```



Apply changes to the new profile

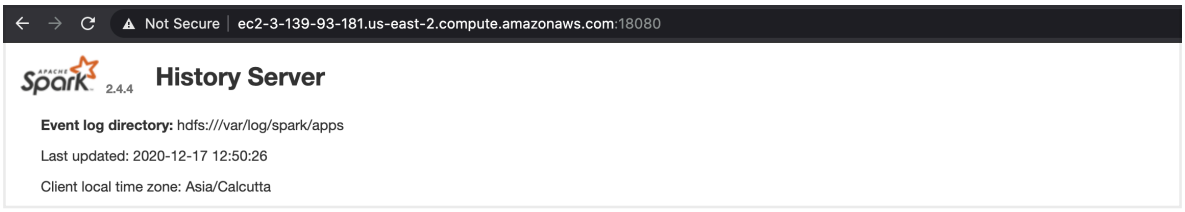
4. Enable the ear-socks-proxy profile



Enable the new SwitchyOmega profile

5. Once, you have configured the proxy, you can access the Spark UI using the command (replace the master node public DNS for you):

```
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.

6. **Reference** - Part 2: Configure Proxy Settings to View Websites Hosted on the Master Node

Terminate Your Cluster to Avoid Excess Billing Charges

Note

- *Do not forget to **Terminate** your EMR cluster after your exercise is finished.*

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

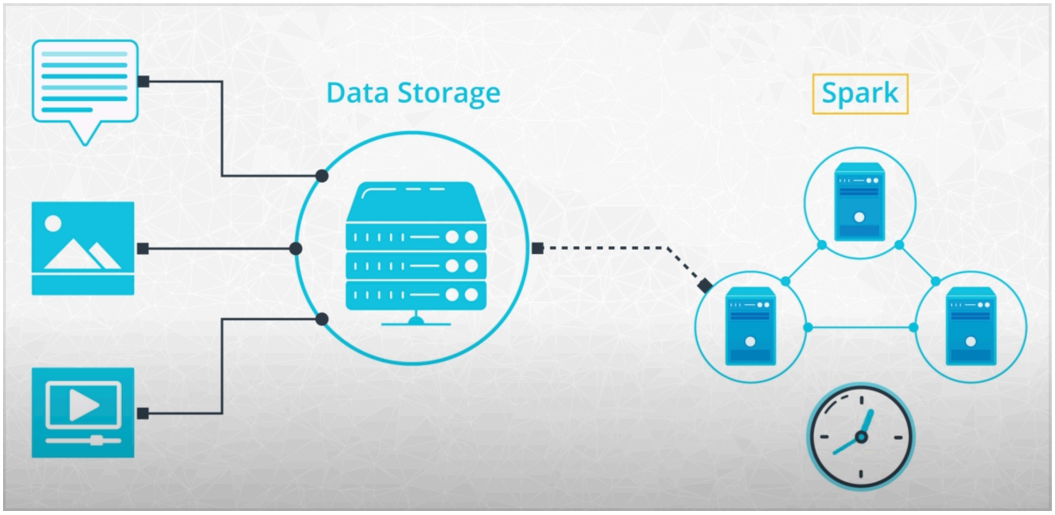
While the use of Jupyter Notebook is common across the industry, you can explore using Zeppelin notebooks. Zeppelin notebooks have been available since EMR 5.x versions, and they have direct access to Spark Context, such as a local spark-shell. For example, if you type sc, you'll be able to get Spark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

Spark Scripts

Jupyter notebooks are great for prototyping as well as exploring and visualizing your data. However, Jupyter notebooks aren't the best tool for automating your workflow, that's where Python scripts come into play.
Imagine you work for a social media company that receives a constant stream of text, image, and video data from your users. To keep up with your users, you want your Spark jobs to run automatically at specific intervals, perhaps once every hour. These Spark jobs give your entire team the latest information on

their users and will probably become so useful to your team that they will start to request more and more Spark jobs. You won't want to manually run a growing collection of notebooks, you'll want automated scripts that you can set and forget. Let's learn how to rework your Jupyter notebooks into Python scripts and submit these scripts to your cluster from the command line.



Running Spark scripts at a time interval

Submitting Spark Script Instructions

Here is the link to the GitHub repo 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:

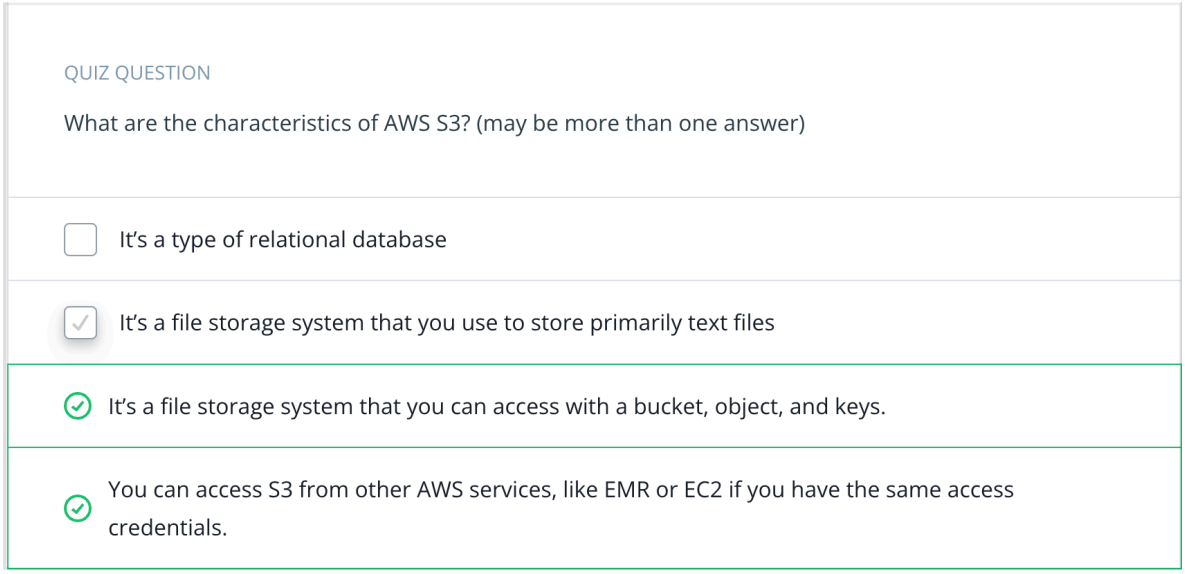
View the Spark UI to understand how your code and workers are working, i.e. which are transformation vs action words (and if they are correctly showing up on Spark UI), and to get familiar with reading the Spark UI. This will give you a better understanding on how your Spark program runs.
Reminder link to Amazon Documentation on Accessing the SparkUI and Amazon documentation on FoxyProxy

Supporting Materials

- Cities

Using S3 to Read and Write Data

We now have a cluster running on AWS and we can submit our Spark programs directly through the command line. How can we read and write data to long-term storage in the cloud?
One of the most common places to store big data sets is Amazon's Simple Storage Service or S3 for short. Amazon S3 is a safe, easy, and cheap place to store big data. Amazon does all the work of maintaining the hardware, keeping backups, and making sure the data is almost always available. You can think about it like Dropbox or iCloud for your big data. You don't need to worry about the details of how S3 works, the Amazon engineers take care of that. You just need to know how to use S3 with Spark. So, next, we will show you how to store data in S3 and then retrieve the data for your Spark program.



Using S3 Buckets to Store Data

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. Imagine you have a structure in S3 like this:

```
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.

Link to Github Repo on Demo code referred to in video

Supporting Materials

- Download Notebook Code and Data: Reading And Writing To AmazonS3

HDFS + SPARK

Differences between HDFS and AWS S3

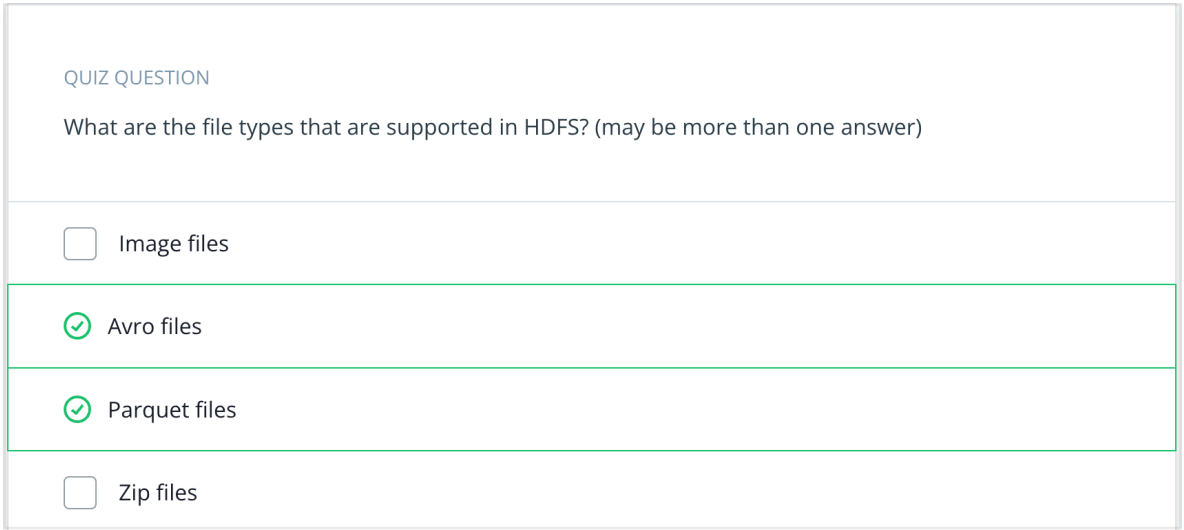
Since Spark does not have its own distributed storage system, it leverages HDFS or AWS S3, or any other distributed storage. Primarily in this course, we will be using AWS S3, but let's review the advantages of using HDFS over AWS S3.

- **AWS S3** is an **object storage system** that stores the data using key value pairs, and **HDFS** is an **actual distributed file system** that guarantees fault tolerance. HDFS achieves fault tolerance by duplicating the same files at 3 different nodes across the cluster by default (it can be configured to reduce or increase this duplication).
- HDFS has traditionally been installed in on-premise systems which had engineers on-site to maintain and troubleshoot the Hadoop Ecosystem, **costing more than storing data in the cloud**. Due to the flexibility of location and reduced cost of maintenance, cloud solutions have been more popular. With the extensive services AWS provides, S3 has been a more popular choice than HDFS.
- Since **AWS S3 is a binary object store**, it can **store all kinds of formats**, even images and videos. HDFS strictly requires a file format - the popular choices are **avro** and **parquet**, which have relatively high compression rates making it useful for storing large datasets.

Reading and Writing to HDFS

In this demo of reading and writing to HDFS, we overview

- SSH into the Spark Cluster
- Viewing the HDFS file system on an EMR Cluster
- Copying JSON files from your local machine to the cluster with the SCP command
- Moving the files into HDFS using HDFS commands



QUIZ QUESTION

Which of these are valid HDFS commands that you can use? (may be more than one answer)

- [ ] `hdfs ls`: shows the files
- [x] `hadoop fs mkdir`: makes a directory
- [ ] `hadoop fs –text` : views raw files
- [x] `hdfs dfs –copyToLocal` : copies files to local (like your laptop)

Lesson Review

Let's recap what you've learned in this lesson.

- You've set up a Spark cluster on AWS
- Submitted Spark code using Jupyter notebooks and Python scripts
- Used distributed data storage using both S3 and HDFS.

So, you now have all the tools to analyze massive datasets on a distributed system. But as you start to use these tools, you'll run into some issues, sometimes in your code and sometimes in your data. In the next lesson, you'll learn how to deal with these issues and debug Spark jobs.