Contents

[Creating an Amazon EMR Cluster 3](#_Toc79670465)

[Setting up Amazon EMR 3](#_Toc79670466)

[Sign up for AWS 3](#_Toc79670467)

[Create an Amazon EC2 key pair for SSH 3](#_Toc79670468)

[Connect to your Linux instance 5](#_Toc79670469)

[Connection options 5](#_Toc79670470)

[Connect to your Linux instance from Windows using PuTTY 5](#_Toc79670471)

[Access Cluster Master using SSH 10](#_Toc79670472)

[Access the Spark Shell 10](#_Toc79670473)

[Prerequisites 10](#_Toc79670474)

[Connect to your Linux instance using an SSH client 10](#_Toc79670475)

[Transfer files to Linux instances using an SCP client 11](#_Toc79670476)

[Configure AWS EMR 14](#_Toc79670477)

[Overview 14](#_Toc79670478)

[Step 1: Plan and configure an Amazon EMR cluster 15](#_Toc79670479)

[Prepare storage for cluster input and output 15](#_Toc79670480)

[Develop and prepare an application for Amazon EMR 15](#_Toc79670481)

[Launch an Amazon EMR cluster 18](#_Toc79670482)

[Step 2: Manage your Amazon EMR cluster 21](#_Toc79670483)

[Submit work to Amazon EMR 21](#_Toc79670484)

[View results 25](#_Toc79670485)

[(Optional) Set up cluster connections 26](#_Toc79670486)

[Step 3: Clean up your Amazon EMR resources 28](#_Toc79670487)

[Shut down your cluster 28](#_Toc79670488)

[Delete S3 resources 30](#_Toc79670489)

[Deleting a Bucket in S3 30](#_Toc79670490)

[Using the S3 console 30](#_Toc79670491)

[Using the AWS CLI 31](#_Toc79670492)

[Exercise 32](#_Toc79670493)

[Clean Up 34](#_Toc79670494)

[Driver Class Config 35](#_Toc79670495)

[Spark Driver and Executor 36](#_Toc79670496)

[Spark Architecture 36](#_Toc79670497)

[DRIVER 36](#_Toc79670498)

[Responsibility of DRIVER 36](#_Toc79670499)

[EXECUTOR 37](#_Toc79670500)

[Responsibility of EXECUTOR 37](#_Toc79670501)

[CLUSTER MANAGER 37](#_Toc79670502)

[Working Process 37](#_Toc79670503)

[Lineage 38](#_Toc79670504)

[How to Configure Executors? 38](#_Toc79670505)

[Spark Caching 39](#_Toc79670506)

[RDD Persistence and Caching in Spark? 39](#_Toc79670507)

[Need of Persistence in Apache Spark 39](#_Toc79670508)

[Benefits of RDD Persistence in Spark 39](#_Toc79670509)

[Storage levels of Persisted RDDs 39](#_Toc79670510)

[MEMORY\_ONLY 39](#_Toc79670511)

[MEMORY\_AND\_DISK 39](#_Toc79670512)

[MEMORY\_ONLY\_SER 40](#_Toc79670513)

[MEMORY\_AND\_DISK\_SER 40](#_Toc79670514)

[DISK\_ONLY 40](#_Toc79670515)

[How to Unpersist RDD in Spark? 40](#_Toc79670516)

[Using an external MySQL database or Amazon Aurora 41](#_Toc79670517)

# Creating an Amazon EMR Cluster

<https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-setting-up.html>

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html#having-ec2-create-your-key-pair>

<https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html>

<https://docs.aws.amazon.com/AmazonS3/latest/userguide/create-bucket-overview.html>

<https://docs.aws.amazon.com/AmazonS3/latest/userguide/uploading-an-object-bucket.html>

## Setting up Amazon EMR

### Sign up for AWS

If you do not have an AWS account, complete the following steps to create one.

**To sign up for an AWS account**

* Open <https://portal.aws.amazon.com/billing/signup>.
* Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

### Create an Amazon EC2 key pair for SSH

To authenticate and connect to the nodes in a cluster over a secure channel using the Secure Shell (SSH) protocol, create an Amazon Elastic Compute Cloud (Amazon EC2) key pair before you launch the cluster. You can also create a cluster without a key pair. This is usually done with transient clusters that start, run steps, and then terminate automatically.

| **If...** | **Then...** |
| --- | --- |
| You already have an Amazon EC2 key pair that you want to use, or you don't need to authenticate to your cluster. | Skip this step. |

#### Create a key pair using Amazon EC2

You can create a key pair using one of the following methods.

**To create your key pair using Console:**

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. In the navigation pane, under **Network & Security**, choose **Key Pairs**.
3. Choose **Create key pair**.
4. For **Name**, enter a descriptive name for the key pair. Amazon EC2 associates the public key with the name that you specify as the key name. A key name can include up to 255 ASCII characters. It can’t include leading or trailing spaces.
5. For **File format**, choose the format in which to save the private key. To save the private key in a format that can be used with OpenSSH, choose **pem**. To save the private key in a format that can be used with PuTTY, choose **ppk**.
6. Choose **Create key pair**.
7. The private key file is automatically downloaded by your browser. The base file name is the name you specified as the name of your key pair, and the file name extension is determined by the file format you chose. Save the private key file in a safe place.

**Important:** This is the only chance for you to save the private key file.

1. If you will use an SSH client on a macOS or Linux computer to connect to your Linux instance, use the following command to set the permissions of your private key file so that only you can read it.

**chmod 400** *my-key-pair***.pem**

If you do not set these permissions, then you cannot connect to your instance using this key pair. For more information, see [Error: Unprotected private key file](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html#troubleshoot-unprotected-key).

**To create your key pair using CLI:**

1. Use the [create-key-pair](https://docs.aws.amazon.com/cli/latest/reference/ec2/create-key-pair.html) AWS CLI command as follows to generate the key and save it to a .pem file.

aws ec2 create-key-pair --key-name *my-key-pair* --query "KeyMaterial" --output text > *my-key-pair*.pem

1. If you will use an SSH client on a macOS or Linux computer to connect to your Linux instance, use the following command to set the permissions of your private key file so that only you can read it.

**chmod 400** *my-key-pair***.pem**

If you do not set these permissions, then you cannot connect to your instance using this key pair. For more information, see [Error: Unprotected private key file](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html#troubleshoot-unprotected-key).

**To create your key pair using Powershell:**

Use the [New-EC2KeyPair](https://docs.aws.amazon.com/powershell/latest/reference/items/New-EC2KeyPair.html) AWS Tools for Windows PowerShell command as follows to generate the key and save it to a .pem file.

PS C:\> (New-EC2KeyPair -KeyName "*my-key*

## Connect to your Linux instance

Connect to the Linux instances that you launched and transfer files between your local computer and your instance.

### Connection options

The operating system of your local computer determines the options that you have to connect from your local computer to your Linux instance.

**If your local computer operating system is Linux or macOS X**

* [SSH client](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html)
* [EC2 Instance Connect](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Connect-using-EC2-Instance-Connect.html)
* [AWS Systems Manager Session Manager](https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html)

**If your local computer operating system is Windows**

* [PuTTY](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html)
* [SSH client](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AccessingInstancesLinux.html)
* [AWS Systems Manager Session Manager](https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html)
* [Windows Subsystem for Linux](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/WSL.html)

### Connect to your Linux instance from Windows using PuTTY

After you launch your instance, you can connect to it and use it the way that you'd use a computer sitting in front of you.

#### Prerequisites

Before you connect to your Linux instance using PuTTY, complete the following prerequisites.

**Verify that the instance is ready**

After you launch an instance, it can take a few minutes for the instance to be ready so that you can connect to it. Check that your instance has passed its status checks. You can view this information in the **Status check** column on the **Instances** page.

**Verify the general prerequisites for connecting to your instance**

To find the public DNS name or IP address of your instance and the user name that you should use to connect to your instance, see [General prerequisites for connecting to your instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html).

**Install PuTTY on your local computer**

Download and install PuTTY from the [PuTTY download page](http://www.chiark.greenend.org.uk/~sgtatham/putty/). If you already have an older version of PuTTY installed, we recommend that you download the latest version. Be sure to install the entire suite.

**Convert your private key using PuTTYgen**

Locate the private key (.pem file) for the key pair that you specified when you launched the instance. Convert the .pem file to a .ppk file for use with PuTTY. For more information, follow the steps in the next section.

#### Convert your private key using PuTTYgen

PuTTY does not natively support the private key format for SSH keys. PuTTY provides a tool named PuTTYgen, which converts keys to the required format for PuTTY. You must convert your private key (.pem file) into this format (.ppk file) as follows in order to connect to your instance using PuTTY.

**To convert your private key**

1. From the **Start** menu, choose **All Programs**, **PuTTY**, **PuTTYgen**.
2. Under **Type of key to generate**, choose **RSA**. If your version of PuTTYgen does not include this option, choose **SSH-2 RSA**.


       RSA key in PuTTYgen
      

1. Choose **Load**. By default, PuTTYgen displays only files with the extension .ppk. To locate your .pem file, choose the option to display files of all types.


       Select all file types
      

1. Select your .pem file for the key pair that you specified when you launched your instance and choose **Open**. PuTTYgen displays a notice that the .pem file was successfully imported. Choose **OK**.
2. To save the key in the format that PuTTY can use, choose **Save private key**. PuTTYgen displays a warning about saving the key without a passphrase. Choose **Yes**.

**Note:** A passphrase on a private key is an extra layer of protection. Even if your private key is discovered, it can't be used without the passphrase. The downside to using a passphrase is that it makes automation harder because human intervention is needed to log on to an instance, or to copy files to an instance.

1. Specify the same name for the key that you used for the key pair (for example, my-key-pair) and choose **Save**. PuTTY automatically adds the .ppk file extension.

Your private key is now in the correct format for use with PuTTY. You can now connect to your instance using PuTTY's SSH client.

#### Connect to your Linux instance

Use the following procedure to connect to your Linux instance using PuTTY. You need the .ppk file that you created for your private key. For more information, see [Convert your private key using PuTTYgen](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html#putty-private-key) in the preceding section. If you receive an error while attempting to connect to your instance, see [Troubleshoot connecting to your instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html).

**To connect to your instance using PuTTY**

1. Start PuTTY (from the **Start** menu, choose **All Programs, PuTTY, PuTTY**).
2. In the **Category** pane, choose **Session** and complete the following fields:
   1. In the **Host Name** box, do one of the following:
      * (Public DNS) To connect using your instance's public DNS name, enter *my-instance-user-name*@*my-instance-public-dns-name*.
      * (IPv6) Alternatively, if your instance has an IPv6 address, to connect using your instance's IPv6 address, enter *my-instance-user-name*@*my-instance-IPv6-address*.

For information about how to get the user name for your instance, and the public DNS name or IPv6 address of your instance, see [Get information about your instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html#connection-prereqs-get-info-about-instance).

* 1. Ensure that the **Port** value is 22.
  2. Under **Connection type**, select **SSH**.


       PuTTY configuration - Session
      

1. (Optional) You can configure PuTTY to automatically send 'keepalive' data at regular intervals to keep the session active. This is useful to avoid disconnecting from your instance due to session inactivity. In the **Category** pane, choose **Connection**, and then enter the required interval in the **Seconds between keepalives** field. For example, if your session disconnects after 10 minutes of inactivity, enter 180 to configure PuTTY to send keepalive data every 3 minutes.
2. In the **Category** pane, expand **Connection**, expand **SSH**, and then choose **Auth**. Complete the following:
   1. Choose **Browse**.
   2. Select the .ppk file that you generated for your key pair and choose **Open**.
   3. (Optional) If you plan to start this session again later, you can save the session information for future use. Under **Category**, choose **Session**, enter a name for the session in **Saved Sessions**, and then choose **Save**.
   4. Choose **Open**.
3. If this is the first time you have connected to this instance, PuTTY displays a security alert dialog box that asks whether you trust the host to which you are connecting.
   1. (Optional) Verify that the fingerprint in the security alert dialog box matches the fingerprint that you previously obtained in [(Optional) Get the instance fingerprint](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html#connection-prereqs-fingerprint). If these fingerprints don't match, someone might be attempting a "man-in-the-middle" attack. If they match, continue to the next step.
   2. Choose **Yes**. A window opens and you are connected to your instance.

**Note:** If you specified a passphrase when you converted your private key to PuTTY's format, you must provide that passphrase when you log in to the instance.

# Access Cluster Master using SSH

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

<https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-ssh-prereqs.html>

## Access the Spark Shell

<https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-shell.html>

### Prerequisites

Before you connect to your Linux instance, complete the following prerequisites.

**Check your instance status**

After you launch an instance, it can take a few minutes for the instance to be ready so that you can connect to it. Check that your instance has passed its status checks. You can view this information in the **Status check** column on the **Instances** page.

**Get the public DNS name and user name to connect to your instance**

To find the public DNS name or IP address of your instance and the user name that you should use to connect to your instance, see [Prerequisites for connecting to your instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html).

**Install an SSH client on your local computer as needed**

Your local computer might have an SSH client installed by default. You can verify this by typing **ssh** at the command line. If your computer doesn't recognize the command, you can install an SSH client.

* Recent versions of Windows Server 2019 and Windows 10 - OpenSSH is included as an installable component. For more information, see [OpenSSH in Windows](https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_overview).
* Earlier versions of Windows - Download and install OpenSSH. For more information, see [Win32-OpenSSH](https://github.com/PowerShell/Win32-OpenSSH/wiki).
* Linux and macOS X - Download and install OpenSSH. For more information, see [https://www.openssh.com](https://www.openssh.com/).

### Connect to your Linux instance using an SSH client

Use the following procedure to connect to your Linux instance using an SSH client. If you receive an error while attempting to connect to your instance, see [Troubleshoot connecting to your instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/TroubleshootingInstancesConnecting.html).

**To connect to your instance using SSH**

1. In a terminal window, use the **ssh** command to connect to the instance. You specify the path and file name of the private key (.pem), the user name for your instance, and the public DNS name or IPv6 address for your instance. For more information about how to find the private key, the user name for your instance, and the DNS name or IPv6 address for an instance, see [Locate the private key](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html#connection-prereqs-private-key) and [Get information about your instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html#connection-prereqs-get-info-about-instance). To connect to your instance, use one of the following commands.
   * (Public DNS) To connect using your instance's public DNS name, enter the following command.

ssh -i */path/my-key-pair*.pem *my-instance-user-name*@*my-instance-public-dns-name*

* + (IPv6) Alternatively, if your instance has an IPv6 address, to connect using your instance's IPv6 address, enter the following command.

ssh -i */path/my-key-pair*.pem *my-instance-user-name*@*my-instance-IPv6-address*

1. You see a response like the following:

The authenticity of host 'ec2-198-51-100-1.compute-1.amazonaws.com (198-51-100-1)' can't be established.

ECDSA key fingerprint is l4UB/neBad9tvkgJf1QZWxheQmR59WgrgzEimCG6kZY.

Are you sure you want to continue connecting (yes/no)?

1. (Optional) Verify that the fingerprint in the security alert matches the fingerprint that you previously obtained in [(Optional) Get the instance fingerprint](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html#connection-prereqs-fingerprint). If these fingerprints don't match, someone might be attempting a "man-in-the-middle" attack. If they match, continue to the next step.
2. Enter **yes**.

You see a response like the following:

Warning: Permanently added 'ec2-198-51-100-1.compute-1.amazonaws.com' (ECDSA) to the list of known hosts.

### Transfer files to Linux instances using an SCP client

One way to transfer files between your local computer and a Linux instance is to use the secure copy protocol (SCP). This section describes how to transfer files with SCP. The procedure is similar to the procedure for connecting to an instance with SSH.

**Prerequisites**

* **Verify the general prerequisites for transferring files to your instance.**

The general prerequisites for transferring files to an instance are the same as the general prerequisites for connecting to an instance. For more information, see [General prerequisites for connecting to your instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html).

* **Install an SCP client**

Most Linux, Unix, and Apple computers include an SCP client by default. If yours doesn't, the OpenSSH project provides a free implementation of the full suite of SSH tools, including an SCP client. For more information, see [https://www.openssh.com](https://www.openssh.com/).

The following procedure steps you through using SCP to transfer a file using the instance's public DNS name, or the IPv6 address if your instance has one.

**To use SCP to transfer files between your computer and your instance**

1. Determine the location of the source file on your computer and the destination path on the instance. In the following examples, the name of the private key file is my-key-pair.pem, the file to transfer is my-file.txt, the user name for the instance is ec2-user, the public DNS name of the instance is my-instance-public-dns-name, and the IPv6 address of the instance is my-instance-IPv6-address.
   * (Public DNS) To transfer a file to the destination on the instance, enter the following command from your computer.

scp -i */path/my-key-pair*.pem */path/my-file.txt* *ec2-user*@*my-instance-public-dns-name*:*path/*

* + (IPv6) To transfer a file to the destination on the instance if the instance has an IPv6 address, enter the following command from your computer. The IPv6 address must be enclosed in square brackets ([ ]), which must be escaped (\).

scp -i */path/my-key-pair*.pem */path/my-file.txt* *ec2-user*@\[*my-instance-IPv6-address*\]:*path/*

1. If you haven't already connected to the instance using SSH, you see a response like the following:

The authenticity of host 'ec2-198-51-100-1.compute-1.amazonaws.com (10.254.142.33)' can't be established.

RSA key fingerprint is 1f:51:ae:28:bf:89:e9:d8:1f:25:5d:37:2d:7d:b8:ca:9f:f5:f1:6f.

Are you sure you want to continue connecting (yes/no)?

(Optional) You can optionally verify that the fingerprint in the security alert matches the instance fingerprint. For more information, see [(Optional) Get the instance fingerprint](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connection-prereqs.html#connection-prereqs-fingerprint).

Enter **yes**.

1. If the transfer is successful, the response is similar to the following:

Warning: Permanently added 'ec2-198-51-100-1.compute-1.amazonaws.com' (RSA) to the list of known hosts.

my-file.txt 100% 480 24.4KB/s 00:00

1. To transfer a file in the other direction (from your Amazon EC2 instance to your computer), reverse the order of the host parameters. For example, you can transfer my-file.txt from your EC2 instance to the a destination on your local computer as my-file2.txt, as shown in the following examples.
   * (Public DNS) To transfer a file to a destination on your computer, enter the following command from your computer.

scp -i */path/my-key-pair*.pem *ec2-user*@*my-instance-public-dns-name*:*path/my-file.txt path/my-file2.txt*

* + (IPv6) To transfer a file to a destination on your computer if the instance has an IPv6 address, enter the following command from your computer. The IPv6 address must be enclosed in square brackets ([ ]), which must be escaped (\).

scp -i */path/my-key-pair*.pem *ec2-user*@\[*my-instance-IPv6-address*\]:*path/my-file.txt path/my-file2.txt*

# Configure AWS EMR

## Overview

With Amazon EMR, you can set up a cluster to process and analyze data with big data frameworks in just a few minutes. This tutorial shows you how to launch a sample cluster using Spark, and how to run a simple PySpark script that you'll store in an Amazon S3 bucket. It covers essential Amazon EMR tasks in three main workflow categories: Plan and Configure, Manage, and Clean Up. You can also adapt this process for your own workloads.


    Workflow diagram for Amazon EMR that outlines the three major workflow
     categories of Plan and Configure, Manage, and Clean Up.
   

**Prerequisites**

* Before you launch an Amazon EMR cluster, make sure you complete the tasks in [Setting up Amazon EMR](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-setting-up.html).

**This tutorial introduces you to the following Amazon EMR tasks:**

**Cost**

* The sample cluster that you create runs in a live environment. The cluster will accrue minimal charges and will only run for the duration of this tutorial as long as you complete the cleanup tasks. Charges accrue at the per-second rate for Amazon EMR pricing and vary by Region. For more information, see [Amazon EMR pricing](https://aws.amazon.com/emr/pricing).
* Minimal charges might also accrue for small files that you store in Amazon S3 for the tutorial. Some or all of your charges for Amazon S3 might be waived if you are within the usage limits of the AWS Free Tier. For more information, see [Amazon S3 pricing](https://aws.amazon.com/s3/pricing) and [AWS Free Tier](https://aws.amazon.com/free/).

## Step 1: Plan and configure an Amazon EMR cluster

In this step, you plan for and launch a simple Amazon EMR cluster with Apache Spark installed. The setup process includes creating an Amazon S3 bucket to store a sample PySpark script, an input dataset, and cluster output.

### Prepare storage for cluster input and output

Create an Amazon S3 bucket to store an example PySpark script, input data, and output data. Create the bucket in the same AWS Region where you plan to launch your Amazon EMR cluster. For example, US West (Oregon) us-west-2. Buckets and folders that you use with Amazon EMR have the following limitations:

* Names can consist of only lowercase letters, numbers, periods (.), and hyphens (-).
* Names cannot end in numbers.
* A bucket name must be unique across all AWS accounts.
* An output folder must be empty.

To create a bucket for this tutorial, see [How do I create an S3 bucket?](https://docs.aws.amazon.com/AmazonS3/latest/user-guide/create-bucket.html) in the Amazon Simple Storage Service Console User Guide.

### Develop and prepare an application for Amazon EMR

In this step, you upload a sample PySpark script to Amazon S3. This is the most common way to prepare an application for Amazon EMR. EMR lets you specify the Amazon S3 location of the script when you submit work to your cluster. You also upload sample input data to Amazon S3 for the PySpark script to process.

We've provided the following PySpark script for you to use. The script processes food establishment inspection data and outputs a file listing the top ten establishments with the most "Red" type violations to your S3 bucket.

**To prepare the example PySpark script for EMR**

1. Copy the example code below into a new file in your editor of choice.
2. Save the file as health\_violations.py.
3. Upload health\_violations.py to Amazon S3 into the bucket you designated for this tutorial. For information about how to upload objects to Amazon S3, see [Uploading an object to a bucket](http://docs.aws.amazon.com/AmazonS3/latest/gsg/PuttingAnObjectInABucket.html) in the Amazon Simple Storage Service Getting Started Guide.

# health\_violations.py

import argparse

from pyspark.sql import SparkSession

def calculate\_red\_violations(data\_source, output\_uri):

"""

Processes sample food establishment inspection data and queries the data to find the top 10 establishments

with the most Red violations from 2006 to 2020.

:param data\_source: The URI where the food establishment data CSV is saved, typically

an Amazon S3 bucket, such as 's3://DOC-EXAMPLE-BUCKET/food-establishment-data.csv'.

:param output\_uri: The URI where the output is written, typically an Amazon S3

bucket, such as 's3://DOC-EXAMPLE-BUCKET/restaurant\_violation\_results'.

"""

with SparkSession.builder.appName("Calculate Red Health Violations").getOrCreate() as spark:

*# Load the restaurant violation CSV data*

if data\_source is not None:

restaurants\_df = spark.read.option("header", "true").csv(data\_source)

*# Create an in-memory DataFrame to query*

restaurants\_df.createOrReplaceTempView("restaurant\_violations")

*# Create a DataFrame of the top 10 restaurants with the most Red violations*

top\_red\_violation\_restaurants = spark.sql("SELECT name, count(\*) AS total\_red\_violations " +

"FROM restaurant\_violations " +

"WHERE violation\_type = 'RED' " +

"GROUP BY name " +

"ORDER BY total\_red\_violations DESC LIMIT 10 ")

*# Write the results to the specified output URI*

top\_red\_violation\_restaurants.write.option("header", "true").mode("overwrite").csv(output\_uri)

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser()

parser.add\_argument(

'--data\_source', help="The URI where the CSV restaurant data is saved, typically an S3 bucket.")

parser.add\_argument(

'--output\_uri', help="The URI where output is saved, typically an S3 bucket.")

args = parser.parse\_args()

calculate\_red\_violations(args.data\_source, args.output\_uri)

**Input arguments**

You must include values for the following arguments when you run the PySpark script as a step.

* --data\_source - The Amazon S3 URI of the food establishment data CSV file. You will prepare this file below.
* --output\_uri - The URI of the Amazon S3 bucket where the output results will be saved.

The input data is a modified version of a publicly available food establishment inspection dataset with Health Department inspection results in King County, Washington, from 2006 to 2020. For more information, see [King County Open Data: Food Establishment Inspection Data](https://data.kingcounty.gov/Health-Wellness/Food-Establishment-Inspection-Data/f29f-zza5). Following are sample rows from the dataset.

name, inspection\_result, inspection\_closed\_business, violation\_type, violation\_points

100 LB CLAM, Unsatisfactory, FALSE, BLUE, 5

100 PERCENT NUTRICION, Unsatisfactory, FALSE, BLUE, 5

7-ELEVEN *#2361-39423A, Complete, FALSE, , 0*

**To prepare the sample input data for EMR**

1. Download the zip file, [food\_establishment\_data.zip](https://docs.aws.amazon.com/emr/latest/ManagementGuide/samples/food_establishment_data.zip).
2. Unzip the content and save it locally as food\_establishment\_data.csv.
3. Upload the CSV file to the S3 bucket that you created for this tutorial. For step-by-step instructions, see [How do I upload files and folders to an S3 bucket?](https://docs.aws.amazon.com/AmazonS3/latest/user-guide/upload-objects.html) in the Amazon Simple Storage Service Console User Guide.

For more information about setting up data for EMR, see [Prepare input data](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-plan-input.html).

### Launch an Amazon EMR cluster

Now that you've completed the prework, you can launch a sample cluster with Apache Spark installed using the latest [Amazon EMR release](https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-release-components.html).

**Note:** If you created your AWS account after December 04, 2013, Amazon EMR sets up a cluster in the [default Amazon Virtual Private Cloud (VPC)](https://docs.aws.amazon.com/vpc/latest/userguide/default-vpc.html) for your selected Region when none is specified.

#### Using console

**To launch a cluster with Spark installed using Quick Options**

1. Sign in to the AWS Management Console and open the Amazon EMR console at <https://console.aws.amazon.com/elasticmapreduce/>.
2. Choose **Create cluster** to open the **Quick Options wizard**.
3. On the **Create Cluster - Quick Options** page, note the default values for **Release**, **Instance type**, **Number of instances**, and **Permissions**. These fields autopopulate with values chosen for general-purpose clusters. For more information about the **Quick Options** configuration settings, see [Summary of Quick Options](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-launch-with-quick-options.html#emr-quick-cluster-options).
4. Enter a **Cluster name** to help you identify the cluster. For example, *My First EMR Cluster*.
5. Leave **Logging** enabled, but replace the **S3 folder** value with the Amazon S3 bucket you created, followed by **/logs**. For example, **s3://DOC-EXAMPLE-BUCKET/logs**. This will create a new folder called 'logs' in your bucket, where EMR will copy the log files of your cluster.
6. Under **Applications**, choose the **Spark** option to install Spark on your cluster. **Quick Options** lets you select from the most common application combinations.

**Note:** It's important to choose the applications that you want to run on your Amazon EMR cluster before you launch the cluster. You can't add or remove applications from a cluster after it has been launched.

1. Under **Security and access**, choose the **EC2 key pair** that you designated or created in [Create an Amazon EC2 key pair for SSH](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-setting-up.html#emr-setting-up-key-pair).
2. Choose **Create cluster** to launch the cluster and open the cluster status page.
3. On the cluster status page, find the **Status** next to the cluster name. The status should change from **Starting** to **Running** to **Waiting** during the cluster creation process. You may need to choose the refresh icon on the right or refresh your browser to receive updates.

When the status progresses to **Waiting**, your cluster is up, running, and ready to accept work.

#### Using CLI

**To launch a cluster with Spark installed using the AWS CLI**

1. Create a Spark cluster with the following command. Enter a name for your cluster with the --name option, and specify the name of your EC2 key pair with the --ec2-attributes option.
2. aws emr create-cluster \
3. --name *"My First EMR Cluster"* \
4. --release-label *emr-5.33.0* \
5. --applications Name=Spark \
6. --ec2-attributes KeyName=*myEMRKeyPairName* \
7. --instance-type m5.xlarge \
8. --instance-count 3 \
9. --use-default-roles

Note the other required values for --instance-type, --instance-count, and --use-default-roles. These values have been chosen for general-purpose clusters. For information about create-cluster used here, see the [AWS CLI reference](https://docs.aws.amazon.com/cli/latest/reference/emr/create-cluster.html).

**Note:** Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

You should see output that includes the ClusterId and ClusterArn of your new cluster. Note your ClusterId, which you will use to check on the cluster status and later to submit work. Following is an example of create-cluster output in JSON format.

{

"ClusterId": "*myClusterId*",

"ClusterArn": "*myClusterArn*"

}

1. Check your cluster status with the following command.
2. aws emr describe-cluster --cluster-id *myClusterId*

You should see output with the Status of your new cluster. Following is an example of describe-cluster output in JSON format.

{

"Cluster": {

"Id": "*myClusterId*",

"Name": "*My First EMR Cluster*",

"Status": {

"State": "STARTING",

"StateChangeReason": {

"Message": "Configuring cluster software"

},

...

},

...

}

{

The status State should change from STARTING to RUNNING to WAITING during the cluster creation process.

When the cluster status progresses to **WAITING**, your cluster is up, running, and ready to accept work.

## Step 2: Manage your Amazon EMR cluster

Now that your cluster is up and running, you can connect to it and manage it. You can also submit work to your running cluster to process and analyze data.

### Submit work to Amazon EMR

With your cluster up and running, you can submit health\_violations.py as a step. A step is a unit of cluster work made up of one or more jobs. For example, you might submit a step to compute values, or to transfer and process data.

You can submit multiple steps to accomplish a set of tasks on a cluster when you create the cluster, or after it's already running. For more information, see [Submit work to a cluster](https://docs.aws.amazon.com/emr/latest/ManagementGuide/AddingStepstoaJobFlow.html).

#### Using Console

**To submit a Spark application as a step using the console**

1. Open the Amazon EMR console at <https://console.aws.amazon.com/elasticmapreduce/>.
2. In **Cluster List**, select the name of your cluster. Make sure the cluster is in a **Waiting** state.
3. Choose **Steps**, and then choose **Add step**.
4. Configure the step according to the following guidelines:
   * For **Step type**, choose **Spark application**. You should see additional fields for **Deploy Mode**, **Spark-submit options**, and **Application location** appear.
   * For **Name**, leave the default value or type a new name. If you have many steps in a cluster, naming each step helps you keep track of them.
   * For **Deploy mode**, leave the default value **Cluster**. For more information about Spark deployment modes, see [Cluster mode overview](https://spark.apache.org/docs/latest/cluster-overview.html) in the Apache Spark documentation.
   * Leave the **Spark-submit options** field blank. For more information about spark-submit options, see [Launching applications with spark-submit](https://spark.apache.org/docs/latest/submitting-applications.html#launching-applications-with-spark-submit).
   * For **Application location**, enter the location of your health\_violations.py script in Amazon S3. For example, *s3://DOC-EXAMPLE-BUCKET/health\_violations.py*.
   * In the **Arguments** field, enter the following arguments and values:
   * --data\_source *s3://DOC-EXAMPLE-BUCKET/food\_establishment\_data.csv*
   * --output\_uri *s3://DOC-EXAMPLE-BUCKET/myOutputFolder*

Replace *s3://DOC-EXAMPLE-BUCKET/food\_establishment\_data.csv* with the S3 URI of the input data you prepared in [Develop and prepare an application for Amazon EMR](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html#emr-getting-started-prepare-app).

Replace *DOC-EXAMPLE-BUCKET* with the name of the bucket you created for this tutorial, and *myOutputFolder* with a name for your cluster output folder.

* + For **Action on failure**, accept the default option **Continue** so that if the step fails, the cluster continues to run.

1. Choose **Add** to submit the step. The step should appear in the console with a status of **Pending**.
2. The status of the step should change from **Pending** to **Running** to **Completed** as it runs. To update the status in the console, choose the refresh icon to the right of the **Filter**. The script takes approximately one minute to run.

You will know that the step finished successfully when the status changes to **Completed**.

#### Using CLI

**To submit a Spark application as a step using the AWS CLI**

1. Make sure you have the ClusterId of the cluster you launched in [Launch an Amazon EMR cluster](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html#emr-getting-started-launch-sample-cluster). You can also retrieve your cluster ID with the following command.

aws emr list-clusters --cluster-states WAITING

1. Submit health\_violations.py as a step with the add-steps command with your ClusterId.
   * You can specify a name for your step by replacing *"My Spark Application"*. In the Args array, replace *s3://DOC-EXAMPLE-BUCKET/health\_violations.py* with the location of your health\_violations.py application.
   * Replace *s3://DOC-EXAMPLE-BUCKET/food\_establishment\_data.csv* with the S3 location of your food\_establishment\_data.csv dataset.
   * Replace *s3://DOC-EXAMPLE-BUCKET/MyOutputFolder* with the S3 path of your designated bucket and a name for your cluster output folder.
   * ActionOnFailure=CONTINUE means the cluster will continue running if the step fails.

aws emr add-steps \

--cluster-id *myClusterId* \

--steps Type=Spark,Name=*"My Spark Application"*,ActionOnFailure=CONTINUE,Args=[*s3://DOC-EXAMPLE-BUCKET/health\_violations.py*,--data\_source,*s3://DOC-EXAMPLE-BUCKET/food\_establishment\_data.csv*,--output\_uri,*s3://DOC-EXAMPLE-BUCKET/MyOutputFolder*]

For more information about submitting steps using the CLI, see the [AWS CLI Command Reference](https://docs.aws.amazon.com/cli/latest/reference/emr/add-steps.html).

After you submit the step, you should see output with a list of StepIds. Since you submitted one step, there should be just one ID in the list. Copy your step ID, which you will use to check the status of the step.

Following is an example of console output in JSON format that you should see after you submit a step.

{

"StepIds": [

"s-1XXXXXXXXXXA"

]

}

1. Query the status of the step with your step ID and the describe-step command. Replace *myClusterId* with your cluster ID.

aws emr describe-step --cluster-id *myClusterId* --step-id *s-1XXXXXXXXXXA*

You should see output with information about your step, as well as a Status section. The following is example describe-step output in JSON format.

{

"Step": {

"Id": "s-1XXXXXXXXXXA",

"Name": "My Spark Application",

"Config": {

"Jar": "command-runner.jar",

"Properties": {},

"Args": [

"spark-submit",

"s3://DOC-EXAMPLE-BUCKET/health\_violations.py",

"--data\_source",

"s3://DOC-EXAMPLE-BUCKET/food\_establishment\_data.csv",

"--output\_uri",

"s3://DOC-EXAMPLE-BUCKET/myOutputFolder"

]

},

"ActionOnFailure": "CONTINUE",

"Status": {

"State": "COMPLETED",

...

}

}

}

The State of the step changes from PENDING to RUNNING to COMPLETED as the step runs. The step takes approximately one minute to run, so you might need to check the status a few times.

You will know that the step was successful when the State changes to **COMPLETED**.

For more information about the step lifecycle, see [Running steps to process data](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview.html#emr-overview-steps).

### View results

After a step runs successfully, you can view its output results in the Amazon S3 output folder you specified when you submitted the step.

**To view the results of health\_violations.py**

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. Choose the **Bucket name** and then the output folder that you specified when you submitted the step. For example, *DOC-EXAMPLE-BUCKET* and then *myOutputFolder*.
3. Verify that the following items are in your output folder:
   * A small-sized object called \_SUCCESS, indicating the success of your step.
   * A CSV file starting with the prefix part-. This is the object with your results.
4. Choose the object with your results, then choose **Download** to save it to your local file system.
5. Open the results in your editor of choice. The output file lists the top ten food establishments with the most Red violations.

Following is an example of health\_violations.py results.

name, total\_red\_violations

SUBWAY, 322

T-MOBILE PARK, 315

WHOLE FOODS MARKET, 299

PCC COMMUNITY MARKETS, 251

TACO TIME, 240

MCDONALD'S, 177

THAI GINGER, 153

SAFEWAY INC *#1508, 143*

TAQUERIA EL RINCONSITO, 134

HIMITSU TERIYAKI, 128

For more information about Amazon EMR cluster output, see [Configure an output location](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-plan-output.html).

### (Optional) Set up cluster connections

This step is not required, but you have the option to connect to cluster nodes with Secure Shell (SSH) for tasks like issuing commands, running applications interactively, and reading log files.

#### Configure security group rules

Before you connect to your cluster, you must set up a Port 22 inbound rule to allow SSH connections.

Security groups act as virtual firewalls to control inbound and outbound traffic to your cluster. When you create a cluster with the default security groups, Amazon EMR creates the following groups:

**ElasticMapReduce-master**

The default Amazon EMR-managed security group associated with the master instance.

**ElasticMapReduce-slave**

The default security group associated with core and task nodes.

**To allow SSH access for trusted sources for the ElasticMapReduce-master security group**

You must first be logged in to AWS as a root user or as an IAM principal that is allowed to manage security groups for the VPC that the cluster is in. For more information, see [Changing Permissions for an IAM User](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_users_change-permissions.html) and the [Example Policy](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_examples_ec2_securitygroups-vpc.html) that allows managing EC2 security groups in the IAM User Guide.

1. Open the Amazon EMR console at <https://console.aws.amazon.com/elasticmapreduce/>.
2. Choose **Clusters**.
3. Choose the **Name** of the cluster.
4. Under **Security and access** choose the **Security groups for Master** link.
5. Choose **ElasticMapReduce-master** from the list.
6. Choose **Inbound**, **Edit**.
7. Check for an inbound rule that allows public access with the following settings. If it exists, choose **Delete** to remove it.
   * **Type**

SSH

* + **Port**

22

* + **Source**

Custom 0.0.0.0/0

**Warning:** Before December 2020, the default EMR-managed security group for the master instance in public subnets was created with a pre-configured rule to allow inbound traffic on Port 22 from all sources. This rule was created to simplify initial SSH connections to the master node. We strongly recommend that you remove this inbound rule and restrict traffic only from trusted sources.

1. Scroll to the bottom of the list of rules and choose **Add Rule**.
2. For **Type**, select **SSH**.

This automatically enters **TCP** for **Protocol** and **22** for **Port Range**.

1. For source, select **My IP**.

This automatically adds the IP address of your client computer as the source address. Alternatively, you can add a range of **Custom** trusted client IP addresses and choose **Add rule** to create additional rules for other clients. Many network environments dynamically allocate IP addresses, so you might need to periodically edit security group rules to update IP addresses for trusted clients.

1. Choose **Save**.
2. Optionally, choose **ElasticMapReduce-slave** from the list and repeat the steps above to allow SSH client access to core and task nodes from trusted clients.

#### Connect to the cluster

After you configure your SSH rules, go to [Connect to the master node using SSH](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-ssh.html) and follow the instructions to:

* Retrieve the public DNS name of the node to which you want to connect.
* Connect to your cluster using SSH.

For more information on how to authenticate to cluster nodes, see [Authenticate to Amazon EMR cluster nodes](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-authenticate-cluster-connections.html).

## Step 3: Clean up your Amazon EMR resources

Now that you've submitted work to your cluster and viewed the results of your PySpark application, you can shut the cluster down and delete your designated Amazon S3 bucket to avoid additional charges.

### Shut down your cluster

Shutting down a cluster stops all of its associated Amazon EMR charges and Amazon EC2 instances.

Amazon EMR retains metadata about your cluster for two months at no charge after you terminate the cluster. This makes it easy to [clone the cluster](https://docs.aws.amazon.com/emr/latest/ManagementGuide/clone-console.html) for a new job or revisit its configuration for reference purposes. Metadata does not include data that the cluster might have written to S3, or that was stored in HDFS on the cluster while it was running.

**Note:** The Amazon EMR console does not let you delete a cluster from the list view after you shut down the cluster. A terminated cluster disappears from the console when Amazon EMR clears its metadata.

#### Using Console

**To shut down the cluster using the console**

1. Open the Amazon EMR console at <https://console.aws.amazon.com/elasticmapreduce/>.
2. Choose **Clusters**, then choose the cluster you want to shut down. For example, *My First EMR Cluster*.
3. Choose **Terminate** to open the **Terminate cluster** prompt.
4. In the open prompt, choose **Terminate** again to shut down the cluster. Depending on the cluster configuration, it may take 5 to 10 minutes to completely terminate and release allocated EC2 resources. For more information about shutting down Amazon EMR clusters, see [Terminate a cluster](https://docs.aws.amazon.com/emr/latest/ManagementGuide/UsingEMR_TerminateJobFlow.html).

**Note:** Clusters are often created with termination protection on to prevent accidental shutdown. If you followed the tutorial closely, termination protection should be off. If termination protection is on, you will see a prompt to change the setting before terminating the cluster. Choose **Change**, then **Off**.

#### Using CLI

**To shut down the cluster using the AWS CLI**

1. Initiate the cluster termination process with the following command, replacing *myClusterId* with the ID of your sample cluster.

aws emr terminate-clusters --cluster-ids *myClusterId*

You should not see any output.

1. To check that the cluster termination process has begun, check the cluster status with the following command.

aws emr describe-cluster --cluster-id *myClusterId*

Following is example output in JSON format. The cluster Status should change from **TERMINATING** to **TERMINATED**. Depending on the cluster configuration, it may take 5 to 10 minutes to completely terminate and release allocated EC2 resources. For more information about shutting down Amazon EMR clusters, see [Terminate a cluster](https://docs.aws.amazon.com/emr/latest/ManagementGuide/UsingEMR_TerminateJobFlow.html)..

{

"Cluster": {

"Id": "j-xxxxxxxxxxxxx",

"Name": "My Cluster Name",

"Status": {

"State": "TERMINATED",

"StateChangeReason": {

"Code": "USER\_REQUEST",

"Message": "Terminated by user request"

},

...

},

...

},

...

}

### Delete S3 resources

Delete the bucket you created earlier to remove all of the Amazon S3 objects used in this tutorial. This bucket should contain your input dataset, cluster output, PySpark script, and log files. You might need to take extra steps to delete stored files if you saved your PySpark script or output in an alternative location.

**Note:** Your cluster must be completely shut down before you delete your bucket. Otherwise, you might run into issues when you try to empty the bucket.

Follow the instructions in [How do I delete an S3 bucket?](https://docs.aws.amazon.com/AmazonS3/latest/user-guide/delete-bucket.html) in the Amazon Simple Storage Service Getting Started Guide to empty your bucket and delete it from S3.

## Deleting a Bucket in S3

### Using the S3 console

**To delete an S3 bucket**

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the **Buckets** list, select the option next to the name of the bucket that you want to delete, and then choose **Delete** at the top of the page.
3. On the **Delete bucket** page, confirm that you want to delete the bucket by entering the bucket name into the text field, and then choose **Delete bucket**.

**Note:** If the bucket contains any objects, empty the bucket before deleting it by selecting the empty bucket configuration link in the **This bucket is not empty** error alert and following the instructions on the **Empty bucket** page. Then return to the **Delete bucket** page and delete the bucket.

### Using the AWS CLI

You can delete a bucket that contains objects with the AWS CLI if it doesn't have versioning enabled. When you delete a bucket that contains objects, all the objects in the bucket are permanently deleted, including objects that are transitioned to the S3 Glacier storage class.

If your bucket does not have versioning enabled, you can use the rb (remove bucket) AWS CLI command with the --force parameter to delete the bucket and all the objects in it. This command deletes all objects first and then deletes the bucket.

$ aws s3 rb s3://bucket-name --force

# Exercise

Run the Estimating Pi Python example from the above link on the cluster: **EstimatePi.py**:

import argparse

import logging

from operator import add

from random import random

from pyspark.sql import SparkSession

logger = logging.getLogger(\_\_name\_\_)

logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(message)s')

def calculate\_pi(partitions, output\_uri):

"""

Calculates pi by testing a large number of random numbers against a unit circle

inscribed inside a square. The trials are partitioned so they can be run in

parallel on cluster instances.

:param partitions: The number of partitions to use for the calculation.

:param output\_uri: The URI where the output is written, typically an Amazon S3

bucket, such as 's3://example-bucket/pi-calc'.

"""

def calculate\_hit(\_):

x = random() \* 2 - 1

y = random() \* 2 - 1

return 1 if x \*\* 2 + y \*\* 2 < 1 else 0

tries = 100000 \* partitions

logger.info(

"Calculating pi with a total of %s tries in %s partitions.", tries, partitions)

with SparkSession.builder.appName("My PyPi").getOrCreate() as spark:

hits = spark.sparkContext.parallelize(range(tries), partitions)\

.map(calculate\_hit)\

.reduce(add)

pi = 4.0 \* hits / tries

logger.info("%s tries and %s hits gives pi estimate of %s.", tries, hits, pi)

if output\_uri is not None:

df = spark.createDataFrame(

[(tries, hits, pi)], ['tries', 'hits', 'pi'])

df.write.mode('overwrite').json(output\_uri)

if \_\_name\_\_ == "\_\_main\_\_":

parser = argparse.ArgumentParser()

parser.add\_argument(

'--partitions', default=2, type=int,

help="The number of parallel partitions to use when calculating pi.")

parser.add\_argument(

'--output\_uri', help="The URI where output is saved, typically an S3 bucket.")

args = parser.parse\_args()

calculate\_pi(args.partitions, args.output\_uri)

Submit a Scala program:

spark-submit --deploy-mode cluster --class org.apache.spark.examples.SparkPi /usr/lib/spark/examples/jars/spark-examples.jar 10

Do it for the Estimate Pi Scala program: **EstimatePi.scala**:

package org.apache.spark.examples

import scala.math.random

import org.apache.spark.\_

*/\*\* Computes an approximation to pi \*/*

object SparkPi {

def main(args: Array[String]) {

val conf = new SparkConf().setAppName("Spark Pi")

val spark = new SparkContext(conf)

val slices = if (args.length > 0) args(0).toInt else 2

val n = math.min(100000L \* slices, Int.MaxValue).toInt *// avoid overflow*

val count = spark.parallelize(1 until n, slices).map { i =>

val x = random \* 2 - 1

val y = random \* 2 - 1

if (x\*x + y\*y < 1) 1 else 0

}.reduce(\_ + \_)

println("Pi is roughly " + 4.0 \* count / n)

spark.stop()

}

}

# Clean Up

<https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-gs.html#emr-getting-started-launch-sample-cluster>

<https://docs.aws.amazon.com/AmazonS3/latest/userguide/delete-bucket.html>

<https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_DeleteInstance.html#USER_DeleteInstance.NoSnapshot>

# Driver Class Config

spark-submit shell script command-line options:

<https://docs.cloudera.com/runtime/7.2.10/running-spark-applications/topics/spark-submit-options.html>

<https://books.japila.pl/apache-spark-internals/tools/spark-submit/#archives>

<https://spark.apache.org/docs/latest/configuration.html>

|  |  |  |
| --- | --- | --- |
| spark.driver.extraClassPath | (none) | Extra classpath entries to prepend to the classpath of the driver.  *Note:* In client mode, this config must not be set through the SparkConf directly in your application, because the driver JVM has already started at that point. Instead, please set this through the --driver-class-path command line option or in your default properties file. |

# Spark Driver and Executor

## Spark Architecture



As we can see that Spark follows Master-Slave architecture where we have one central coordinator and multiple distributed worker nodes. The central coordinator is called Spark Driver and it communicates with all the Workers.

Each Worker node consists of one or more Executor(s) who are responsible for running the Task. Executors register themselves with Driver. The Driver has all the information about the Executors at all the time.

This working combination of Driver and Workers is known as Spark Application.

The Spark Application is launched with the help of the Cluster Manager. Cluster manager can be any one of the following –

1. Spark Standalone Mode
2. YARN
3. Mesos
4. Kubernetes

## **DRIVER**

Driver is a Java process. This is the process where the main() method of our Scala, Java, Python program runs. It executes the user code and creates a SparkSession or SparkContext and the SparkSession is responsible to create DataFrame, DataSet, RDD, execute SQL, perform Transformation & Action, etc.

### **Responsibility of DRIVER**

1. The main() method of our program runs in the Driver process. It creates SparkSession or SparkContext.
2. Conversion of the user code into Task (transformation and action). It looks at the user code and determines are the possible Tasks, i.e., the number of tasks to be performed is decided by the Driver.  
   But how does it determine the Tasks? – With the help of Lineage.
3. Helps to create the Lineage, Logical Plan and Physical Plan.
4. Once the Physical Plan is generated, the Driver schedules the execution of the tasks by coordinating with the Cluster Manager.
5. Coordinates with all the Executors for the execution of Tasks. It looks at the current set of Executors and schedules our tasks.
6. Keeps track of the data (in the form of metadata) which was cached (persisted) in Executor’s (worker’s) memory.

## **EXECUTOR**

Executor resides in the Worker node. Executors are launched at the start of a Spark Application in coordination with the Cluster Manager.

They are dynamically launched and removed by the Driver as per required.

### **Responsibility of EXECUTOR**

1. To run an individual Task and return the result to the Driver.
2. It can cache (persist) the data in the Worker node.

Thinking how these Driver and Executor Processes are launched after submitting a job (spark-submit)?  
Well, then let’s talk about the Cluster Manager.

## **CLUSTER MANAGER**

Spark is dependent on the Cluster Manager to launch the Executors and also the Driver (in Cluster mode).

We can use any of the Cluster Manager (as mentioned above) with Spark i.e. Spark can be run with any of the Cluster Manager.

Spark provides a script named “**spark-submit**” which helps us to connect with a different kind of Cluster Manager and it controls the number of resources the application is going to get i.e. it decides the number of Executors to be launched, how much CPU and memory should be allocated for each Executor, etc.

### **Working Process**

spark-submit –master <Spark master URL> –executor-memory 2g –executor-cores 4 WordCount-assembly-1.0.jar

1. Let’s say a user submits a job using “spark-submit”.
2. “spark-submit” will in-turn launch the Driver which will execute the main() method of our code.
3. Driver contacts the cluster manager and requests for resources to launch the Executors.
4. The cluster manager launches the Executors on behalf of the Driver.
5. Once the Executors are launched, they establish a direct connection with the Driver.
6. The driver determines the total number of Tasks by checking the Lineage.
7. The driver creates the Logical and Physical Plan.
8. Once the Physical Plan is generated, Spark allocates the Tasks to the Executors.
9. Task runs on Executor and each Task upon completion returns the result to the Driver.
10. Finally, when all Task is completed, the main() method running in the Driver exits, i.e. main() method invokes sparkContext.stop().
11. Finally, Spark releases all the resources from the Cluster Manager.

## Lineage

When a new RDD is derived from an existing RDD using transformation, that new RDD contains a pointer to the parent RDD and Spark keeps track of all the dependencies between these RDDs using a component called the Lineage. In case of data loss, this lineage is used to rebuild the data. DataFrame, DataSet, SQL are internally converted to RDDs for computation as RDDs are the lowest level of abstraction in Spark. So, all the transformations that are involved internally in a DataFrame, DataSet, SQL can be seen by converting them to RDD.

## How to Configure Executors?

In spark-defaults.conf:

spark.executor.instances 4

In code:

spark.conf.set("spark.executor.instances", 4)

spark.conf.set("spark.executor.cores", 4)

In code, setting the parameters "spark.executor.instances" and "spark.executor.cores" on the SparkConf object.

SparkConf conf = new SparkConf()

// 4 executor per instance of each worker

.set("spark.executor.instances", "4")

// 5 cores on each executor

.set("spark.executor.cores", "5");

With the spark-submit command:

./bin/spark2-submit \

--master yarn \

--deploy-mode cluster \

--driver-memory 8g \

--executor-memory 16g \

--executor-cores 2 \

--num-executors 4

--class org.apache.spark.examples.SparkPi \

/spark-home/examples/jars/spark-examples\_versionxx.jar 80

Use:

--class org.apache.spark.examples.SparkPi /usr/lib/spark/examples/jars/spark-examples.jar 10

# Spark Caching

## RDD Persistence and Caching in Spark?

Spark**RDD**persistence is an optimization technique in which saves the result of RDD evaluation. Using this we save the intermediate result so that we can use it further if required. It reduces the computation overhead.  
We can make persisted RDD through **cache()** and **persist()** methods. When we use the cache() method we can store all the RDD in-memory. We can persist the RDD in memory and use it efficiently across parallel operations.  
The difference between cache() and persist() is that using cache() the default storage level is **MEMORY\_ONLY** while using persist() we can use various storage levels (described below). It is a key tool for an interactive algorithm. Because, when we persist RDD each node stores any partition of it that it computes in memory and makes it reusable for future use. This process speeds up the further computation ten times.  
When the RDD is computed for the first time, it is kept in memory on the node. The cache memory of the Spark is fault tolerant so whenever any partition of RDD is lost, it can be recovered by transformation Operation that originally created it.

## Need of Persistence in Apache Spark

In Spark, we can use some RDD’s multiple times. If honestly, we repeat the same process of**RDD** **evaluation** each time it required or brought into action. This task can be time and memory consuming, especially for iterative algorithms that look at data multiple times. To solve the problem of repeated computation the technique of persistence came into the picture.

## Benefits of RDD Persistence in Spark

There are some advantages of RDD caching and persistence mechanism in spark. It makes the whole system

* Time efficient
* Cost efficient
* Lessen the execution time.

## Storage levels of Persisted RDDs

Using **persist()** we can use various storage levels to Store Persisted RDDs in Apache Spark. Let’s discuss each RDD storage level one by one-

### MEMORY\_ONLY

In this storage level, RDD is stored as deserialized Java object in the JVM. If the size of RDD is greater than memory, It will not cache some partition and recompute them next time whenever needed. In this level the space used for storage is very high, the CPU computation time is low, the data is stored in-memory. It does not make use of the disk.

### MEMORY\_AND\_DISK

In this level, RDD is stored as deserialized Java object in the JVM. When the size of RDD is greater than the size of memory, it stores the excess partition on the disk, and retrieve from disk whenever required. In this level the space used for storage is high, the CPU computation time is medium, it makes use of both in-memory and on disk storage.

### MEMORY\_ONLY\_SER

This level of Spark store the RDD as serialized Java object (one-byte array per partition). It is more space efficient as compared to deserialized objects, especially when it uses fast serializer. But it increases the overhead on CPU. In this level the storage space is low, the CPU computation time is high and the data is stored in-memory. It does not make use of the disk.

### MEMORY\_AND\_DISK\_SER

It is similar to **MEMORY\_ONLY\_SER**, but it drops the partition that does not fits into memory to disk, rather than recomputing each time it is needed. In this storage level, The space used for storage is low, the CPU computation time is high, it makes use of both in-memory and on disk storage.

### DISK\_ONLY

In this storage level, RDD is stored only on disk. The space used for storage is low, the CPU computation time is high and it makes use of on disk storage..

## How to Unpersist RDD in Spark?

Spark monitor the cache of each node automatically and drop out the old data partition in the LRU (least recently used) fashion. LRU is an algorithm which ensures the least frequently used data. It spills out that data from the cache. We can also remove the cache manually using **RDD.unpersist()** method.

# Using an external MySQL database or Amazon Aurora

To use an external MySQL database or Amazon Aurora as your Hive metastore, you override the default configuration values for the metastore in Hive to specify the external database location, either on an Amazon RDS MySQL instance or an Amazon Aurora instance.

**Note:** Hive neither supports nor prevents concurrent write access to metastore tables. If you share metastore information between two clusters, you must ensure that you do not write to the same metastore table concurrently, unless you are writing to different partitions of the same metastore table.

The following procedure shows you how to override the default configuration values for the Hive metastore location and start a cluster using the reconfigured metastore location.

**To create a metastore located outside of the EMR cluster**

1. Create a MySQL or Aurora database.

For information about how to create an Amazon RDS database, see <https://aws.amazon.com/rds/>.

1. Modify your security groups to allow JDBC connections between your database and the **ElasticMapReduce-Master** security group.

For information about how to modify your security groups for access, see [Working with Amazon EMR-managed security groups](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-man-sec-groups.html).

1. Set JDBC configuration values in **hive-site.xml**:

**Important:** If you supply sensitive information, such as passwords, to the Amazon EMR configuration API, this information is displayed for those accounts that have sufficient permissions. If you are concerned that this information could be displayed to other users, create the cluster with an administrative account and limit other users (IAM users or those with delegated credentials) to accessing services on the cluster by creating a role which explicitly denies permissions to the elasticmapreduce:DescribeCluster API key.

Create a configuration file called hiveConfiguration.json containing edits to hive-site.xml as shown in the following example.

<*hostname*> is the DNS address of the Amazon RDS instance running the database. <*username*> and <*password*> are the credentials for your database. For more information about connecting to MySQL and Aurora database instances, see [Connecting to a DB instance running the MySQL database engine](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToInstance.html) and [Connecting to an Athena DB cluster](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Connect.html) in the Amazon RDS User Guide. javax.jdo.option.ConnectionURL is the JDBC connect string for a JDBC metastore. javax.jdo.option.ConnectionDriverName is the driver class name for a JDBC metastore.

The MySQL JDBC drivers are installed by Amazon EMR.

The value property can not contain any spaces or carriage returns. It should appear all on one line.

[

{

"Classification": "hive-site",

"Properties": {

"javax.jdo.option.ConnectionURL": "jdbc:mysql://*hostname*:3306/hive?createDatabaseIfNotExist=true",

"javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver",

"javax.jdo.option.ConnectionUserName": "username",

"javax.jdo.option.ConnectionPassword": "password"

}

}

]

Reference the hiveConfiguration.json file when you create the cluster as shown in the following AWS CLI command. In this command, the file is stored locally, you can also upload the file to Amazon S3 and reference it there, for example, s3://mybucket/hiveConfiguration.json.

**Note**: Linux line continuation characters (\) are included for readability. They can be removed or used in Linux commands. For Windows, remove them or replace with a caret (^).

aws emr create-cluster --release-label *emr-5.33.0* --instance-type m5.xlarge --instance-count 2 \

--applications Name=Hive --configurations file://hiveConfiguration.json --use-default-roles

1. Connect to the master node of your cluster.

For information about how to connect to the master node, see [Connect to the master node using SSH](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-connect-master-node-ssh.html) in the Amazon EMR Management Guide.

1. Create your Hive tables specifying the location on Amazon S3 by entering a command similar to the following:

CREATE EXTERNAL TABLE IF NOT EXISTS table\_name

(

key int,

value int

)

LOCATION s3://*mybucket/hdfs*/

1. Add your Hive script to the running cluster.

Your Hive cluster runs using the metastore located in Amazon RDS. Launch all additional Hive clusters that share this metastore by specifying the metastore location.