Dataproc (/dataproc) and Apache Spark (https://spark.apache.org/) provide infrastructure and capacity that you can use to run Monte Carlo simulations written in Java, Python, or Scala.

Monte Carlo methods can help answer a wide range of questions in business, engineering, science, mathematics, and other fields. By using repeated random sampling to create a probability distribution for a variable, a Monte Carlo simulation can provide answers to questions that might otherwise be impossible to answer. In finance, for example, pricing an equity option requires analyzing the thousands of ways the price of the stock could change over time. Monte Carlo methods provide a way to simulate those stock price changes over a wide range of possible outcomes, while maintaining control over the domain of possible inputs to the problem.

In the past, running thousands of simulations could take a very long time and accrue high costs. Dataproc enables you to provision capacity on demand and pay for it by the minute. Apache Spark lets you use clusters of tens, hundreds, or thousands of servers to run simulations in a way that is intuitive and scales to meet your needs. This means that you can run more simulations more quickly, which can help your business innovate faster and manage risk better.

Security is always important when working with financial data. Dataproc runs on Google Cloud, which helps to keep your data safe, secure, and private (/security/overview) in several ways. For example, all data is encrypted during transmission and when at rest, and Google Cloud is ISO 27001, SOC3, and PCI compliant (/security/compliance).

- Create a managed Dataproc cluster with Apache Spark pre-installed (/dataproc/docs/concepts/dataproc-versions#supported_cloud_dataproc_versions).
- Run a Monte Carlo simulation using Python that estimates the growth of a stock portfolio over time.
- Run a Monte Carlo simulation using Scala that simulates how a casino makes money.

This tutorial uses the following billable components of Google Cloud:

- Compute Engine (/compute/pricing)

- Cloud Storage (/storage/pricing)
- Dataproc (/dataproc/docs/resources/pricing)

To generate a cost estimate based on your projected usage, use the pricing calculator (/products/calculator) . New Google Cloud users might be eligible for a free trial (/free-trial).

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. For more information, see Cleaning up (#clean-up).

- Set up a Google Cloud project
    1. Sign in (https://accounts.google.com/Login) to your Google Account.

       If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).
    2. In the Cloud Console, on the project selector page, select or create a Cloud project.

       ★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

       Go to the project selector page (https://console.cloud.google.com/projectselector2/home/dashboard)
    3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (/billing/docs/how-to/modify-project).
    4. Enable the Dataproc and Compute Engine APIs.

       Enable the APIs (https://console.cloud.google.com/flows/enableapi?apiid=dataproc,compute_component)
    5. Install and initialize the Cloud SDK (/sdk/docs/).
- Create a Cloud Storage bucket (/storage/docs/xml-api/put-bucket-create) in your Cloud project
    1. In the Cloud Console, go to the **Cloud Storage Browser** page.

       Go to the Cloud Storage Browser page (https://console.cloud.google.com/storage/browser)
    2. Click **Create bucket**.
    3. In the **Create bucket** dialog, specify the following attributes:

- A unique bucket name, subject to the bucket name requirements (/storage/docs/bucket-naming#requirements).
- A storage class (/storage/docs/storage-classes).
- A location where bucket data will be stored.

4. Click **Create**.

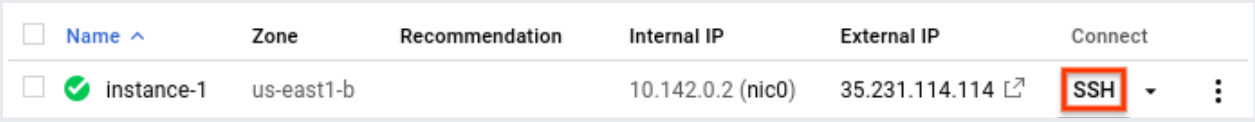
Follow the steps to create a Dataproc cluster (/dataproc/docs/guides/create-cluster#creating_a_cloud_dataproc_cluster) from the Google Cloud Console. The default cluster settings, which includes two-worker nodes, is sufficient for this tutorial.

By default, Apache Spark prints verbose logging in the console window. For the purpose of this tutorial, change the logging level to log only errors. Follow these steps:

1. In the Cloud Console, go to the **VM instances** page.

   Go to the VM instances page (https://console.cloud.google.com/compute/instances)

2. In the list of virtual machine instances, click **SSH** in the row of the instance that you want to connect to.



A browser window opens at your home directory on the primary node.

1. From the primary node's home directory, edit `/etc/spark/conf/log4j.properties`.

2. Set `log4j.rootCategory` equal to `ERROR`.

3. Save the changes and exit the editor. If you want to enable verbose logging again, reverse the change by restoring the value for `.rootCategory` to its original (`INFO`) value.

Spark supports Python, Scala, and Java as programming languages for standalone applications, and provides interactive interpreters for Python and Scala. The language you choose is a matter of personal preference. This tutorial uses the interactive interpreters because you can experiment by changing the code, trying different input values, and then viewing the results.

In finance, Monte Carlo methods are sometimes used to run simulations that try to predict how an investment might perform. By producing random samples of outcomes over a range of probable market conditions, a Monte Carlo simulation can answer questions about how a portfolio might perform on average or in worst-case scenarios.

Follow these steps to create a simulation that uses Monte Carlo methods to try to estimate the growth of a financial investment based on a few common market factors.

Note: This code is provided only as an example. Don't use this code to make investment decisions.

1. Start the Python interpreter from the Dataproc primary node.

Wait for the Spark prompt `>>>`.

2. Enter the following code. Make sure you maintain the indentation in the function definition.

3. Press `return` until you see the Spark prompt again.

   The preceding code defines a function that models what might happen when an investor has an existing retirement account that is invested in the stock market, to which they add additional money each year. The function generates a random return on the investment, as a percentage, every year for the duration of a specified term. The function takes a seed value as a parameter. This value is used to reseed the random number generator, which ensures that the function doesn't get the same list of random numbers each time it runs. The `random.normalvariate` function ensures that random values occur across a normal distribution (https://wikipedia.org/wiki/Normal_distribution) for the specified mean and standard deviation. The function increases the value of the portfolio by the growth amount, which could be positive or negative, and adds a yearly sum that represents further investment.

   You define the required constants in an upcoming step.

4. Create many seeds to feed to the function. At the Spark prompt, enter the following code, which generates 10,000 seeds:

   The result of the `parallelize` operation is a resilient distributed dataset (RDD) (http://spark.apache.org/docs/1.0.1/programming-guide.html#resilient-distributed-datasets-rdds), which is a

collection of elements that are optimized for parallel processing. In this case, the RDD contains seeds that are based on the current system time.

When creating the RDD, Spark slices the data based on the number of workers and cores available. In this case, Spark chooses to use eight slices, one slice for each core. That's fine for this simulation, which has 10,000 items of data. For larger simulations, each slice might be larger than the default limit. In that case, specifying a second parameter to `parallelize` can increase the number slices, which can help to keep the size of each slice manageable, while Spark still takes advantage of all eight cores.

5. Feed the RDD that contains the seeds to the growth function.

   The `map` method passes each seed in the RDD to the `grow` function and appends each result to a new RDD, which is stored in `results`. Note that this operation, which performs a *transformation*, doesn't produce its results right away. Spark won't do this work until the results are needed. This *lazy evaluation* is why you can enter code without the constants being defined.

6. Specify some values for the function.

7. Call `reduce` to aggregate the values in the RDD. Enter the following code to sum the results in the RDD:

8. Estimate and display the average return:

   Be sure to include the dot (`.`) character at the end. It signifies floating-point arithmetic.

9. Now change an assumption and see how the results change. For example, you can enter a new value for the market's average return:

10. Run the simulation again.

11. When you're done experimenting, press `CTRL+D` to exit the Python interpreter.

Monte Carlo, of course, is famous as a gambling destination. In this section, you use Scala to create a simulation that models the mathematical advantage that a casino enjoys in a game of chance. The "house edge" at a real casino varies widely from game to game; it can be over 20% in keno (https://wikipedia.org/wiki/Keno), for example. This tutorial creates a simple game where the house has only a one-percent advantage. Here's how the game works:

- The player places a bet, consisting of a number of chips from a bankroll fund.
- The player rolls a 100-sided die (how cool would that be?).
- If the result of the roll is a number from 1 to 49, the player wins.
- For results 50 to 100, the player loses the bet.

You can see that this game creates a one-percent disadvantage for the player: in 51 of the 100 possible outcomes for each roll, the player loses.

Follow these steps to create and run the game:

1. Start the Scala interpreter from the Dataproc primary node.

2. Copy and paste the following code to create the game. Scala doesn't have the same requirements as Python when it comes to indentation, so you can simply copy and paste this code at the `scala>` prompt.

3. Press `return` until you see the `scala>` prompt.

4. Enter the following code to play the game 25 times, which is the default value for `NUMBER_OF_GAMES`.

   Your bankroll started with a value of 10 units. Is it higher or lower, now?

5. Now simulate 10,000 players betting 100 chips per game. Play 10,000 games in a session. This Monte Carlo simulation calculates the probability of losing all your money before the end of the session. Enter the follow code:

   Note that the syntax `.reduce(_+_)` is shorthand in Scala for aggregating by using a summing function. It is functionally equivalent to the `.reduce(add)` syntax that you saw in the Python example.

   The preceding code performs the following steps:

   - Creates an RDD with the results of playing the session.
   - Replaces bankrupt players' results with the number `1` and nonzero results with the number `0`.
   - Sums the count of bankrupt players.
   - Divides the count by the number of players.

   A typical result might be:

   Which represents a near guarantee of losing all your money, even though the casino had only a one-percent advantage.

To avoid incurring charges to your Google Cloud Platform account for the resources used in this tutorial:

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an **`appspot.com`** URL, delete selected resources inside the project instead of deleting the whole project.

1. In the Cloud Console, go to the **Manage resources** page.

   Go to the Manage resources page (https://console.cloud.google.com/iam-admin/projects)

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

- For more on submitting Spark jobs to Dataproc without having to use `ssh` to connect to the cluster, read Dataproc—Submit a job (/dataproc/docs/guides/submit-job)
- Try out other Google Cloud features for yourself. Have a look at our tutorials (/docs/tutorials).