Four Easy Steps to Optimize Your Code on Cloud Resources

Whether it be data collection, ingestion, engineering, resource creation, automation, or other, being able to write code optimally is very important for time and budget management of cloud resources. Now consider all the places where you have code running in the background - think Lambda functions, Cloud9 scripts, CodeBuild, or SageMaker notebooks. Are you ever wishing they would run faster, be more organized, use less memory, be easier to reproduce or cost less? Think of all the money you could spend on other security resources or time you could spend providing other security solutions if you didn't have to waste it on these issues. All you need to do is follow these four main rules when performing operational excellence on any solution written in a coding language. They've been tried and tested on our graph implementation tool, where we re-apply each of these solutions for purposes of operational excellence every sprint.

Functionalized Code

Have you ever started writing a script where you end up copying and pasting a few lines over and over again, changing some variables here and there, until you end up on line 1150 dazed and confused? That's because you are not writing your code in a functionalized manner. To remedy this, define a function for the lines of code that are repetitive, and pass the variables in every time you call that function. This allows you to stay organized, increase reproduction ease, and speed up the coding process.

Object Oriented Programming

Once you get skilled at being a functionalized programmer, you can further increase your organization by implementing object-oriented programming (OOP). OOP can be thought of as a file system for your functions. You can think of each function as a 'file'. Then, you can think of a class as a 'directory' full of files. You can think of a module as the 'directory of all directories' for a certain subject. Once you understand that organization, you can reference any 'file' from any 'directory' as long as you refer to the 'directory of all directories'. In other words, you can reference any function from one class as long as you import the module that the classes (and functions) are defined in. So, OOP allows you to split up your functionalized code into many different python scripts (or modules), keeping your entire solution extremely organized and reusable for future projects that could use similar functions.

Multiprocessing

Okay, so we have all of our code extremely well organized, but it is still taking forever to run. We can use multiprocessing as a tool to reduce this time. Multiprocessing allows you to run multiple functions at once. The amount of functions you can apply this to is unlimited, but the number of processes at a time is dependent on your number of CPUs. For example, if you have 4 CPUs available to you and you have 4 different functions that take the same amount of time to run, you can use multiprocessing to reduce a runtime of 4 seconds to a runtime of 1 second. What if I had 5 functions, the first one takes 1 second and the last four of them take 2 seconds. The first CPU will complete the first function in one second while the other 3 take on the next three functions, which will complete in two seconds. The final

function will be assigned to the next open CPU (or the first CPU in this case). So the entire multi-process will take 3 seconds (the first function + the last function). Multiprocessing can be really helpful when wanting to reduce runtime which also correlates to cost reduction when using CodeBuild, SageMaker, etc. In our case, we were able to reduce our runtime by 88% after implementing multiprocessing for the first time. Just wait until you see what it can do.

Variable Deletion to Reduce Memory Usage

Another big cost saver is memory management. The more memory required to run your code, the higher the costs to have a resource running it. This is especially apparent when working with big data. An easy solution to this is just deleting any variables that are housing large data frames or lists when you are done using them. When applying this during data engineering for our graph solution, we were able to reduce our memory usage by ~50%.

Measure this using CloudWatch

Don't forget about CloudWatch. You can measure memory usage, CPU usage, and runtimes with Cloudwatch. It provides tools such as Dashboards, Alarms, and Alerts to keep you the most up to date and notified of your current code environments. You can monitor how applying the above steps improved your solution in every aspect, so don't wait to take advantage of this easy to use tool.

I hope these four steps can help make your code reproducible, speed up your code runtime and lower your cloud resource costs. I know that they have helped me a ton when working on the graph solution, so I am super thankful I could share some of these tips to you all. Please don't hesitate to reach out if you have any questions or just want to talk shop!

Sources used:

- https://towardsdatascience.com/why-developers-are-falling-in-love-with-functional-programming-13514df4048e
- https://www.educative.io/blog/object-oriented-programming
- https://realpython.com/python3-object-oriented-programming/
- https://www.geeksforgeeks.org/multiprocessing-python-set-1/
- https://stackabuse.com/basics-of-memory-management-in-python/