

GEOG 596B – Individual Studies – Capstone Project

**Final Project Report**

Advisor: James O’Brien

By Edirlei Santos

*April 2020*

Considerations on deploying geoprocessing workflows as serverless functions

# Goal

This capstone project seeks to understand the possible benefits of employing serverless cloud technology to run geoprocessing intensive tasks.

# Introduction

Cloud technology provides a remarkable quantity of computer resources in an economic model that allows ordinary institutions to have access to large amount of computing power. As a consequence, cloud resources empower new business or research opportunities to emerge from those who otherwise could not afford to acquire and to manage advanced computing technology. In this context, GIS professionals can legitimately inquire about the potential advantages in using powerful cloud resources to improve geoprocessing workflows usually performed with desktop computers.

When I learned to use AWS EC2 I was impressed by how much processing power it was possible to configure in a cloud computing environment. When FaaS came to my attention, my initial assumption was that serverless functions had similar processing capabilities. This being the case, I though serverless could be a good option for GIS analysts to test their geoprocessing code. In particular, the geoprocessing code that take long time to execute, like image analysis and geostatistics.

Moreover, cloud resources like network, storage type, memory, and processor, can be granularly configured. On the other side, the demand for computer resources in geoprocessing vary depending on type of task being performed, which in my experience it is usually storage input/output or processor. For this reason, there was the idea of bumping up only the required cloud resource for the task at hand and establish a more optimal computing environment.

Given these points, with the serverless benefits of no server to configure and with the cloud resources potential, I proposed to explore serverless as a platform to run geoprocessing and avoid possible slowdown of running it on local/on-premise computers.

## Potential benefits from using cloud computing

The proper use of cloud resources can be advantageous in contrast to using on-premises computer infrastructure. This understanding provides a background to the possibilities that motivates this research. For this reason, it is appropriate to enumerate some advantages that are relevant to my study. They include:

1. The significant reduced time in planning, purchasing, provisioning, and configuring new servers. Furthermore, on-premise hardware tends towards obsolescence over time.
2. The elasticity characteristic of cloud computing allows optimal use of hardware resources. Resources automatically scale through time according to demand.
3. The engineers that setup cloud infrastructure are among the best qualified for each specific area of responsibility. In other words, the use of cloud computers enables customers to have highly qualified professionals working on their server infrastructure, a factor that can make the cloud a decisive advantage for many corporations.

## Function as a Service

Cloud resources are offered as layers of services, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). They mostly differ based on what portion is managed by the cloud provider that sells the service and what portion is managed by the one buying the service. For this study I will be exploring the Function as a Service (FaaS) layer, also referred to as “serverless”. For further reading about cloud technology I suggest the book The Cloud at Your Service by Mateos and Rosenberg (2011).

FaaS is a relatively recent service offering from cloud providers. Its architecture removes much of the need for a traditional always-on server component (Fowler, 2019). In this architecture the user uploads the software code, called function, into the serverless framework. The elementary state of the function is inactive, taking no memory or processor resources. The execution of the function will only happen after an event happens and calls the function, then the cloud provider dynamically allocates the computing resources to execute the code. Once the execution of the function is completed the computer resources are deallocated. Accordingly, some of the FaaS characteristics are:

1. Typical users are developers.
2. Users do not need to worry about designing and maintaining server infrastructure.
3. Costs are related only to computing resources allocated for the execution of the code.
4. Function execution is restricted to a time duration – for example, AWS Lambda is 15 minutes.
5. The events that trigger functions are defined by the cloud provider. Typical events include file storage change, scheduled tasks, and https type requests.
6. Because there is no server setup, FaaS reduces the time that takes from completed code to the time the code is deployed into production.

## FaaS and Relevant Notes from Literature Review

The move from coding based on a server architecture to Serverless is a contemporary trend (Bala & Vincent, 2019) . One of the reasons serverless has been popular is because developers can focus more on the code and less on the infrastructure dependencies (Fenty, 2018). FaaS is the straightforward way for developers to adopt the cloud and it might well become the unifying platform for cloud applications. Not surprisingly, the benefits for developers using FaaS are improved productivity, greater IT satisfaction, improved ability to innovate, greater flexibility, and faster development cycles (Hendrickson et al., 2016).

Despite its popularity, according to Spillner, Mateos, and Monge (2017), FaaS is gaining traction primarily in three areas. First, in the Internet-of-Things applications where connected devices emit data sporadically. Second, for web applications with light-weight backend tasks. Third, as a coupling mechanism between other cloud computing services.

Spillner (2017) remarks that FaaS services lack specialized options for scientific computing. In his paper he describes FaaS performance for processing for mathematics, computer graphics, cryptology and meteorology. In the computer graphics performance topic, that closely relates to GIS image analysis, the conclusion was that because of intense I/O the performance only becomes competitive when 19 or more threads are used – this assessment was using AWS S3 object storage, Python 2 at AWS Lambda in the year 2017. Thus, to improve performance, he suggests looking at developing multithreading in the function code and using in-memory I/O.

# Research justification

“FaaS is a relatively new concept that was first made available in 2014 by hook.io and is now implemented in services such as AWS Lambda and Google Cloud Function” (Han, 2017). Accordingly, the amount of academic research related to this technology is small. By the same token, the amount of research that relates GIS and serverless is scarce. Zugliani (2018) notes that publications about serverless are targeted to developers rather than researchers. As a consequence, research that explores the serverless potential in geospatial sciences is well desired. In contrast to the serverless commercial popularity, not much work has been done to explore its potential for scientific and high-performance computing applications with more demanding execution requirements.

Indeed, this is the case with some GIS operations that require a large amount of processing power; for example, geostatistics and imagery analysis. In such operations, if the target data is large, processing time is substantial. Since cloud technology is now prevalent, it is relevant to consider utilizing cloud resources to run GIS processor intensive tasks. Resorting to cloud hosted servers provides several benefits, which have already been previously mentioned. Still, GIS analysts, the ones interested in spatial data analysis, would have to deal with the complexity of setting up cloud infrastructure, or to depend on the overhead of an IT group to do so. On the other hand, FaaS is a cloud offer that removes the need for setting up servers and it gives GIS analysts the autonomy to readily manage the resource.

Moreover, when geoprocessing needs occur infrequently, another benefit of using FaaS is that the function only uses processing resources when it is being executed. This is a compelling model from an economical point of view – using cloud resources without having to deal with the cost of a server that need to run continuously.

Therefore, the purpose of this capstone project comes from the need to understand the use of FaaS for geoprocessing tasks, what makes the outcome of this project a welcome addition to the GIS research domain. Indeed, this understanding can be used as a reference for GIS professionals inquiring about improving similar geoprocessing workflows.

# Research Methodology

This research work intends to be fundamental and exploratory in its purpose (Wikipedia, 2019). The expected outcome is a description of possible advantages and disadvantages of using FaaS as a platform to deploy geoprocessing functions.

## Research Questions

1. Are serverless platform, a recommended service resource to run geoprocessing workflows?
2. Is there a more recommended option for geoprocessing among the providers Amazon AWS, Microsoft Azure and Google Cloud?

The experiments of function development and function deployment that was done for this research were done using AWS Lambda. The observations on this project document about the Microsoft Azure Functions and the Google Cloud Functions are an inference based on the documentation and online articles from the respective providers.

# Experiments with AWS Lambda

To assess the worthiness of using FaaS for geoprocessing, the original plan was to compare the run time of an image analysis code between its execution using an ordinary modern desktop computer and its execution using a serverless function. My attempt to do this experiment was done using AWS Lambda and the ArcGIS API Python for package. Soon, I realized the limitation of the Lambda max deployment package size. To explain, this package includes essential libraries used by the ArcGIS package, like NumPy. However, it also has libraries that are not relevant in a serverless deployment, like the Jupyter libraries. So, I removed directories of some obvious non-essential packages and then I was able to deploy a test function that used my reduced version of the ArcGIS API. This was just a test trick; the ideal would be to systematically determine and isolate only the necessary modules to use for each function. Nevertheless, the max package size is always a factor to consider in functions that need additional libraries, like it is usually the case with geoprocessing code.

The next step on my experiment was to run geoprocessing workflow on an image stored on the AWS S3. The workflow I tested was a watershed model (Support, 2020) using a small DEM (about 10 square miles) and it took more time than I expected: 11 minutes; what was close to the limit of 15 minutes. To my disappointment, the available setting for the AWS Lambda memory/processing settings was at its maximum. Since my intent was to run the image analysis on a larger image, I resorted to explore the “divide and conquer” idea that is described in the next paragraph.

Because each Lambda function execution has a constraining small processing power, there was the idea to use the serverless distributed computing capability. The plan was to use an image processing type where the image can be divided into smaller segments for individual processing and the individual results be assembled into one image that is the same as if the original image were processed undivided. This way, I could make concurrent calls to the Lambda function where the inputs are smaller segments of a larger image what would reduce the overall processing time. The findings of this plan are described in the upcoming section below “Geoprocessing Considerations”.

## FaaS features from AWS, Azure and Google Cloud and related tools

It is worthy to compare and contrast the FaaS specifications from prominent cloud providers. This way, we can have an overview of what is generally available, and also know what a more adequate provider can be if a required feature is differing among the other providers. Table 1 describes the variations among the cloud providers in implementing serverless features that are relevant to geoprocessing deployment (Makai, 2020) (AWS, 2020a; Google, 2020; Microsoft, 2020)

Table . FaaS offerings comparison

|  |  |  |  |
| --- | --- | --- | --- |
|  | AWS | Azure Functions | Google Cloud Functions |
| Triggers | S3, Amazon DynamoDB, API Gateway, Alexa, AWS IoT, Cron events, Amazon SNS, Amazon CloudWatch and more. | Storage, CosmosDB, Event Grid, HTTP requests, IoT Hub, Timer, Service Bus, Twilio (SMS) (no trigger from log service yet) | HTTP, Cloud Storage, Pub/Sub (message service), Firestore, Firebase |
| Logs and Monitoring | AWS CloudWatch and X-Ray | Application Insights and Azure Monitor Logs | By using Cloud Pub/Sub |
| Authoring functionsTools | Specialized integration with AWS Cloud 9. Plugins for Visual Studio Code and PyCharm. | Visual Studio Code extension | No equivalent found |
| Development tools | AWS Serverless Application Model (SAM). Needs docker for local testing. Open source options: Chalice, Serverless Framework, Zappa. | Azure Functions Core Tools. Needs docker for local testing. | gcloud – generic command line framework to manage Google Cloud Platform Resourcesno reference to local testing |
| Publishing Functions | SAM, S3 or zip file | Remote build using requirements.txt (preferred) | Using requirements.txt Packaging local dependencies |
| Stack Resource Creation | SAM, CloudFormation | Generic to Azure resources, not specialized as AWS SAM Azure Pipeline Azure Resource Manager | Sparse references to “Google Cloud Deployment Manager” with yaml format |
| Function timeout | 15 minutes | 10 minutes (a premium plan allows unlimited duration) | Default 1 minute, max is 9 minutes |
| Max Concurrent execution | 1000 | 200 | 1000 |
| Deployment package size limit | 250MB | Azure storage is required to host the function. Ranges from 1G to 1TB | 500MB |
| Temp storage | 512MB | 1G to 1TB | Limit not mentioned |
| Max Memory allocation | Up to 3008MB | 1500MB | 2048MB |
| Security  | Base security at default and SAM templates allow easy configuration | Developer needs to plan security | Developer needs to plan security |

Not difficult to observe on Table 1 that each provider allocates the computing resources differently. This divergence reveals a prevailing lack of standards for serverless implementation. Some of the differences reflect how the function accesses the other cloud services each provider offers. For example, AWS Lambda does not have a direct http trigger, instead one has to use another AWS service for this purpose, the AWS Gateway - on the other side, the AWS Gateway is in many aspects a much more capable service, what might make it worth the extra work.

From my point of view, what the providers offer in the features of logs and monitoring, publishing functions, stack resource creation and memory allocation, are generally equivalent, and usually these would not be a defining factor to prefer one among the other providers.

The AWS Lambda has better options for triggers, authoring functions tools, development tools and security. I describe more about AWS tools further ahead.

Azure Functions is a good provider option for function timeout, deployment package size limit and temp storage. Interesting to note, under the Azure Functions it is possible to have an unlimited function time and generous storage for the function. High-demanding geoprocessing workflows seems possible on Azure, but likely with disappointing execution time due to the low processing power and low I/O speed.

The Google Cloud Functions does not seem to have any distinguishing feature. Now, I did like how the Google Cloud documentation is thoughtfully written. It clearly describes the concepts that relate to the serverless architecture. Even if I utilize the services from another serverless cloud provider, I could use some of the Google Cloud Functions documentation to understand a serverless concept.

All three platforms appear to be capable of hosting Python geoprocessing tasks with similar limitations. From the perspective of a GIS analyst, AWS offers easier and more sophisticated options to develop and deploy a function. Azure and Google require specific knowledge of professional developer tools and methods to develop and publish functions. This observation is outlined in the next section.

# Tools for Serverless Development

There is a good amount of developer tools that facilitates serverless development. For this project I looked at tools that are relatively simple to setup and deploy code to serverless and can relate to GIS analysts that create geoprocessing code. Notoriously, it is generally easier to find online resources and answers that are related to the AWS platform (AWS, 2020c), and some tools that can be a resource for serverless development are mentioned below.

### Serverless Framework

The Serverless Framework is an MIT open-source project that aims to provide FaaS workflow automation and best practices. An interesting aspect of the Serverless Framework is that it is capable of generating YAML deployment code for many cloud providers (Source, 2020). At the moment, it seems to be the leading attempt to standardize serverless deployment across different cloud providers. For those not using AWS Lambda to host their functions, the Serverless Framework is recommended. For those utilizing AWS Lambda the Serverless Framework is a great option, at the same time, the AWS fully fledged development tools offer more breadth and depth to leverage the AWS resources.

### AWS SAM

“The AWS Serverless Application Model (SAM) is an open-source framework for building serverless applications.”(AWS, 2020b). When AWS SAM is installed in a computer it provides a docker container that simulates the AWS Lambda, so coding can be developed and tested on the developer’s local computer. Also, it has a simple syntax to create supplemental AWS components for the functions, such as Gateway and S3. AWS SAM can be used with any of the Lambda compatible programming languages.

### AWS Chalice

Chalice is a microframework for writing serverless apps in Python.(Saryerwinnie, 2020). The Chalice functionality is similar to the AWS SAM; however it is specific to Python. I used Chalice to deploy my test serverless functions and it was remarkably easy to install and use. It is worthy to mention that Python development and deployment can be troubling to deal with packages dependencies. Chalice can automate the packages needed for deployment when the file requirements.txt is set accordingly. For further information on Python packaging refer to Foundation (2020).

### AWS Boto3

Boto3 is an AWS Python package that facilitates the integration with the Python application and the AWS services including Amazon S3, Amazon EC2, Amazon DynamoDB, and more. For developers looking for an efficient way to use cloud resources from within the Python serverless functions code, Boto3 gives AWS an edge when compared other providers. There is no equivalent option on Azure or Google Cloud.

Considerations for geoprocessing deployment on FaaS

## Resource Limitations

Overall, when deploying geoprocessing functions, I observed a few limitations that are important to know before planning these types of functions. The findings are:

* The function lifetime limit is an issue for processes that have long execution times.
* FaaS platforms have modest maximum processing power in contrast to the Platform as a Service model where processing and memory can be set to relatively high maximums.
* Potential bottlenecks when high I/O bandwidth is required.
* The deployment package and memory limits are relatively low. The ArcGIS Python module, for example, has 314GB installed with dependencies. Hopefully in the future streamlined smaller packages are made available for serverless.
* Function portability between cloud providers is not straightforward.

## Geoprocessing considerations

Geoprocessing that does not entail neighborhood operations requires little processing power. This type of operation can easily be executed in local machines. The use of serverless in this case is useful in deployment scenarios that services a multiclient architecture.

On the other hand, geoprocessing that involves neighborhood operations can demand increased processing power and could benefit from using cloud processing. However, because serverless platforms have a relatively low maximum processing power, the possible route would be to divide the raster into smaller segments and place individual requests to AWS Lambda for each segment, and then later join the resulting outputs. This way, if processing an image raster for example, the total raster processing time is theoretically divided by the number of existing segments. However there is a fundamental problem with neighborhood operations because these operations need all cells present in the same process to produce a correct result. Dividing a raster into smaller segments to process and join the individual outputs later will produce an invalid result.

## Prospects and challenges

My initial assertion before starting this research work was that FaaS could be a cloud service option for highly demanding geoprocessing. However, my geoprocessing function deployment attempts have demonstrated that serverless has insufficient computing resources for processor intensive cases.

To have a more practical experience, I searched for modular geoprocessing codes, the ones that can clearly be a function, in order to develop them as serverless. The good candidates I found are related to light-weight geoprocessing, which was not my primary focus. Nevertheless, I conceptualized this light-weight geoprocessing code in a serverless framework, and it was then that I had the enlightening understanding that these cases greatly benefit from the serverless distributed software architecture. So, my original assumption was misinformed and intensive geoprocessing on serverless is not recommended. On the other side, I found that some lightweight geoprocessing workflows in a distributed scenario is possible and, more importantly, it has extraordinary application potential and it is an opportunity for innovators.

With this understanding, it is valuable to mention some prospects for geoprocessing with serverless. Some geoprocessing activities that could take advantage of serverless include: small batch geocoding, IoT spatiotemporal data gathering, pixel based supervised image classification, mobile fleet tracking, seismometers, emergency feeds (FEMA, Earth Observatory Natural Event Tracker (EONET)), connected cars, location-based crime records plus live AI neighborhood camera systems, agricultural IoT (like humidity sensors), asset tracking etc. Indeed, these examples are a suggestion from this study about what are the practical geoprocessing projects worthwhile to pursue by using serverless technology.

The challenging scenarios relate to spatial analysis that require neighborhood operations, like satellite imagery object detection and geostatistical simulations of large datasets. As serverless technologies evolve and imagery resolution increases, it may then be reasonable to develop techniques that could allow development of distributed systems that can process neighborhood dependent analyses of spatial data in a way that larger datasets can be partitioned into smaller ones; achieving the same result as processing the spatial data in one unit.

Moreover, a noteworthy aspect is that serverless platforms prefer that its input/output, when possible, be coded as RESTful APIs. This can benefit geoprocessing workflows when it abstracts the underlying Python language, allowing other programming languages to consume the result of serverless functions. As a consequence, geoprocessing services can be easily integrated into dissimilar software platforms and expand GIS opportunities. Admittedly, this fact is also a suggestion from this study about prospective development of geoprocessing workflows using serverless.

Also, when there is more than on GIS analyst working on a geoprocessing workflow, using serverless for a team environment is advantageous because the serverless architecture is inherently decentralized. The geoprocessing code development, tests, and deployment in serverless can happen in parallel. Code deployment advantages in the serverless scenario is a research topic on its own. From my findings, for GIS analysts that use Python, I suggest the use of a Git client of choice plus the Serverless Framework(Source, 2020).

## Containers as an option

Those doing geoprocessing with ArcGIS have the option of taking advantage of the available docker container referred to as ArcGIS Notebook Server. This option is particularly well suited for those doing GIS exploratory analysis. However, it is possible to have deployment scenarios where code execution is triggered by events. Yet, additional packages are necessary and, possibly, the development of components and orchestration objects outside the container to handle security, data storage, traffic, logs etc. Docker containers require a more complex infrastructure setup compared to serverless but are still a leaner solution than having a full server system setup and running. Also, since a docker is always on, the pricing model differs from FaaS.

# Concluding thoughts

FaaS offers a flexible new architectural model and the benefits are still being discovered. The simplicity of serverless entails a rapid pace of innovation. Consequently, it is worthy to consider serverless for geoprocessing. “the benefits of not needing to own, scale, manage, or secure infrastructure lets developers focus on their business instead of the care and feeding of servers or containers.” (Wagner, 2019)

To answer the research question “are serverless platforms, a recommended service resource to run geoprocessing workflows?”, it is wise to say that for now the recommended geoprocessing workflows using serverless are the lightweight processing that take advantage of distribute scenarios. The key to determine if a geoprocessing workflow can make good use of serverless has less to do with the cloud processing capabilities, and it has more to do with the understanding about each particular geoprocessing code fitness for the serverless inherently distributed computing architecture.

Regarding the research question “is there a more recommended option for geoprocessing among the providers Amazon AWS, Microsoft Azure and Google Cloud?”, they offer similar overall serverless processing capabilities. However, AWS documentation is more prominent, and my understanding is that AWS Lambda has better developer tools.

One question to keep in mind, “is serverless a chance for geoprocessing developers to start thinking about workflows to move from homogeneous deployment to a more distributed architecture?” The considerations raised in this study suggest that the benefits of deploying geoprocessing workflows on serverless can potentially outweigh the cost of a transition.

The serverless architecture offers an unprecedented opportunity for GIS analysts working with geoprocessing, not only to focus more on their solution but also to create new pathways to deploy them. Serverless extends the geoprocessing landscape into innovation, and potentially a revolution.

# References

AWS. (2020a). AWS Lambda Limits. Retrieved from AWS Lambda Limits

AWS. (2020b). AWS Serverless Application Model. Retrieved from <https://aws.amazon.com/serverless/sam/>

Bala, R., & Vincent, P. (2019). The Atomization of IT. 7.

Fenty, K. (2018). *Serverless Platforms Unlock the Full Benefit of Microservices and Cloud*. Retrieved from

Foundation, T. P. S. (2020). PIP user guide. *Read the Docs*. Retrieved from <https://pip.readthedocs.io/en/latest/user_guide/#requirements-files>

Fowler, M. (2019). Serverless Architectures. Retrieved from <https://martinfowler.com/articles/serverless.html>

Google. (2020). Google Cloud Function Quotas. Retrieved from <https://cloud.google.com/functions/quotas>

Han, B. (2017). An Introduction to Serverless and FaaS. Retrieved from [https://medium.com/@BoweiHan/an-introduction-to-serverless-and-faas-functions-as-a-service-fb5cec0417b2](https://medium.com/%40BoweiHan/an-introduction-to-serverless-and-faas-functions-as-a-service-fb5cec0417b2)

Hendrickson, S., Sturdevant, S., Harter, T., Venkataramani, V., Arpaci-Dusseau, A. C., & Arpaci-Dusseau, R. H. J. E. (2016). Serverless computation with openlambda. *60*, 80.

Makai, M. (2020). Azure Functions. *Full Stack Python*. Retrieved from <https://www.fullstackpython.com/azure-functions.html>

Mateos, A., & Rosenberg, J. (2011). The Cloud at Your Service. *Helion, West Midlands*.

Microsoft. (2020). Azure Functions Python Code Examples. Retrieved from <https://docs.microsoft.com/en-us/samples/browse/?products=azure-functions&languages=python>

Saryerwinnie, J. (2020). Chalice. *Read the Docs*. Retrieved from <https://chalice.readthedocs.io/en/latest/>

Source, S. F. O. (2020). Retrieved from <https://serverless.com/>

Spillner, J., Mateos, C., & Monge, D. A. (2017). *Faaster, better, cheaper: The prospect of serverless scientific computing and hpc.* Paper presented at the Latin American High Performance Computing Conference.

Support, E. T. (2020). How To: Create a watershed model using the Hydrology toolset. Retrieved from <https://support.esri.com/en/technical-article/000012346>

Wagner, T. (2019). The Serverless Supercomputer. *A Cloud Guru*. Retrieved from <https://read.acloud.guru/https-medium-com-timawagner-the-serverless-supercomputer-555e93bbfa08>

Wikipedia. (2019). Exploratory Research. Retrieved from <https://en.wikipedia.org/wiki/Exploratory_research>

Zugliani, M. (2018). Serverless data processing applied to big mobility data.