Lambda at scale: Tips & Tricks to make your Serverless application scalable

9 December 2022 - 7 min. read

AWS Lambda | Serverless

NOTE: All quotas in this article refer to the date on which the article has been written. They could change over time.

Introduction

Serverless, serverless and serverless... If you are familiar with our blog, this is one of the words we like the most.

It allows you to focus entirely on the application code, forget about your infrastructure, and pay only for the computational power used.

However, there are many aspects to consider when it comes to serverless computing to avoid unpleasant surprises (perhaps in production). Your application must not only be cloud-native but also serverless-native.

This article will give you many hints and tips to keep in mind when developing an application on Lambda at scale.

So let's summarize some aspects to remember when working with Lambda on large numbers.

Pay attention to the concurrency

When working on Lambda, one of the first limits to keep in mind is the maximum concurrency per region, 1000 by default. This means there can be a maximum of 1000 simultaneous Lambda executions for each AWS region. This limit is shared across all Lambda instances, being a region-level constraint.

Therefore, if you have several workloads in the same region, it could be useful to configure reserved concurrency for each workload to set a maximum cap for each of them and avoid behaviors that are difficult to analyze and resolve.

In this way, the various workloads will not "steal" the availability of Lambda if ever one of these must scale suddenly.

In many cases, the limit of 1000 is not enough. Fortunately, this limit is *soft,* so it can be increased directly from the AWS console up to a few thousand. Remember that you often have to "justify" many quota increases to AWS.

And if that's still not enough? The organization of your AWS accounts might need a little overhaul; maybe you need to properly configure an AWS Organization and create AWS accounts for each workload and each environment.

If you would like to refresh some concepts about provisioned concurrency, reserved concurrency, and cold start time, please take a look at this previous article, in which these aspects of Lambda are analyzed in detail. https://www.proud2becloud.com/a-comprehensive- analysis-of-aws-lambda-function-optimize-spikes-and-prevent-cold-starts/

All the suggestions in this article derive from having increased the Lambda limit from 1000 to 10,000 for one of our production applications. The increase itself is straightforward but involves a series of possible side effects.

API calls to external services (S3 and Secrets Manager)

Most likely, our application will have to call external services.

For example, among the various best practices (not only from AWS) is that of not hard-coding sensitive data in the application code, but saving them on appropriate

services, perhaps encrypted at rest and in transit, managing access policies in a granular way.

For example, let's talk about Secrets Manager and Parameter Store. Have you ever tried to see their quotas?

With Secrets Manager, we are pretty safe since the quota to find a secret is 10,000 per second, but not for Parameter Store, since it is 40 per second. The limit can be increased, but it must be taken into consideration before being released into production.

Did you know that it is also possible to cache the values of our secrets? Thanks to **Lambda Extensions**, it is possible to extend the functionality of our Lambdas for monitoring, observability, and governance purposes. It is possible to use extensions developed by other partners to integrate directly with them, for example, to forward application logs.

There are also software packages for Parameter Store and Secret Manager that allow us to cache the various parameters and secrets without having to do it in the Lambda code.

Would you like to see how Lambda Extensions work? Let us know!

Maybe our Lambda might also have to download a file from S3. Have you ever read the quotas of this service? For writing and reading data, it has very particular limits, i.e. 3500 PUT/COPY/POST/DELETE and 5000 GET/HEAD requests per second, per **partitioned prefix**.

So this limit is not shared at the account or region level but applies to each bucket. We could make a separate article if you are interested, but know that you could run into this limit if you have to download a single file many times from Lambda.

Tip: if you have to download the same file many times (thousands of times in a few seconds), try copying the file to different partitions.

In conclusion, keep attention to the quotas of all the services we integrate with.

File system & /tmp storage

What if we need to access a file system during the Lambda computing phase? For example, if we have to download data from S3 to analyze its content? Luckily, in Lambda, you can use the /tmp mounting point for this operation.

Obviously, it is not persistent. When the Lambda returns "cold" AWS will care about emptying that portion of storage so that it is free for the next execution.

This limit has been increased recently. Before, you only had 512 MB. (Now it can be increased up to 10GB). If you needed more space, you could hook up an EFS by putting the Lambda in VPC.

What have we discovered? In the rapid and massive scaling phase, the Lambda may not necessarily have this portion of empty storage, so make sure you clean it up if you have to do any operations, or you could find old data!

Database connection

Most likely, our Lambda function will not only have to call external services or write something to the storage, but it will also have to connect to a database to perform read and write operations. Have you ever tried to open 5000 connections temporarily to a production database? :D

To avoid overloading the database with operations of this type, we recommend using a connection pool and interfacing with it. AWS offers RDS Proxy for this type of operation. Therefore, you can forget about managing the connection pool with the database. AWS will take care of it with a managed service.

Serverless application at scale

What do we remember from this experience? Constantly evaluate the metrics of your application in advance to check if they fit with what you have designed.

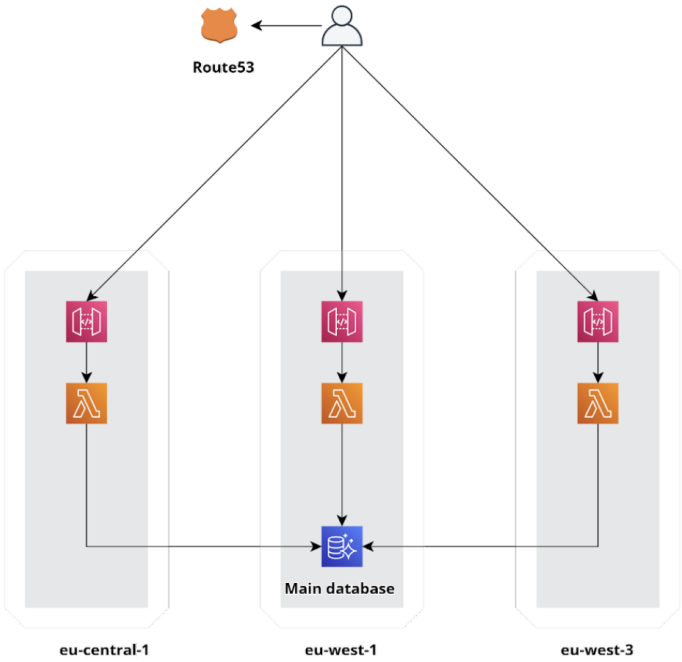
What if you don't have previous metrics to use, as in our case? By following the Operational Excellence AWS Well-Architect Framework's pillar, we could readapt and evolve the new infrastructure in a very short time.

Unfortunately, the numbers in production could not fit well for an infrastructure released in a single region. Therefore, we took advantage of the global infrastructure

of AWS by deploying the infrastructure in different Regions and doing load balancing between them.

Let's add some cross-region routing with Route 53, and our global infrastructure is ready!

So know that if your workload exceeds the limits in only one region, it may be time to go Global! Luckily on AWS, it is very simple; if we make use of the Infrastructure as Code (IaC) paradigm, it will be a matter of a few clicks!



AWS re:Invent 2022 pro tip: SnapStart

We just came back from the AWS re:Invent (don't worry, the article about it will be out soon!), where we first heard about an exciting feature for Lambda: SnapStart.

This feature is only available for Java, as it aims to address this engine's known high cold start time issues.

The way it works is quite simple; a snapshot is made for each version of the Lambda, and it will be used in subsequent executions.

It is possible to use special hooks to instantiate the AWS SDK client, whose time is known for Java, and perhaps save some secrets or information that would normally be collected during the Lambda startup phase.

To conclude

AWS offers us a unique global infrastructure in terms of functionality and reliability, as well as a series of "building blocks" to create Serverless, high-performance, and cost-effective infrastructures. However, we must design everything so that the application is scalable even on large numbers, if necessary.

Let us know if you liked this article and want to learn more about some of the topics mentioned!

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Alessandro Bertini

DevOps Engineer @ beSharp. I deal with Cloud-Native software development, strongly oriented to the serverless paradigm!Passionate about board games and video games (as the best geeks do!)
