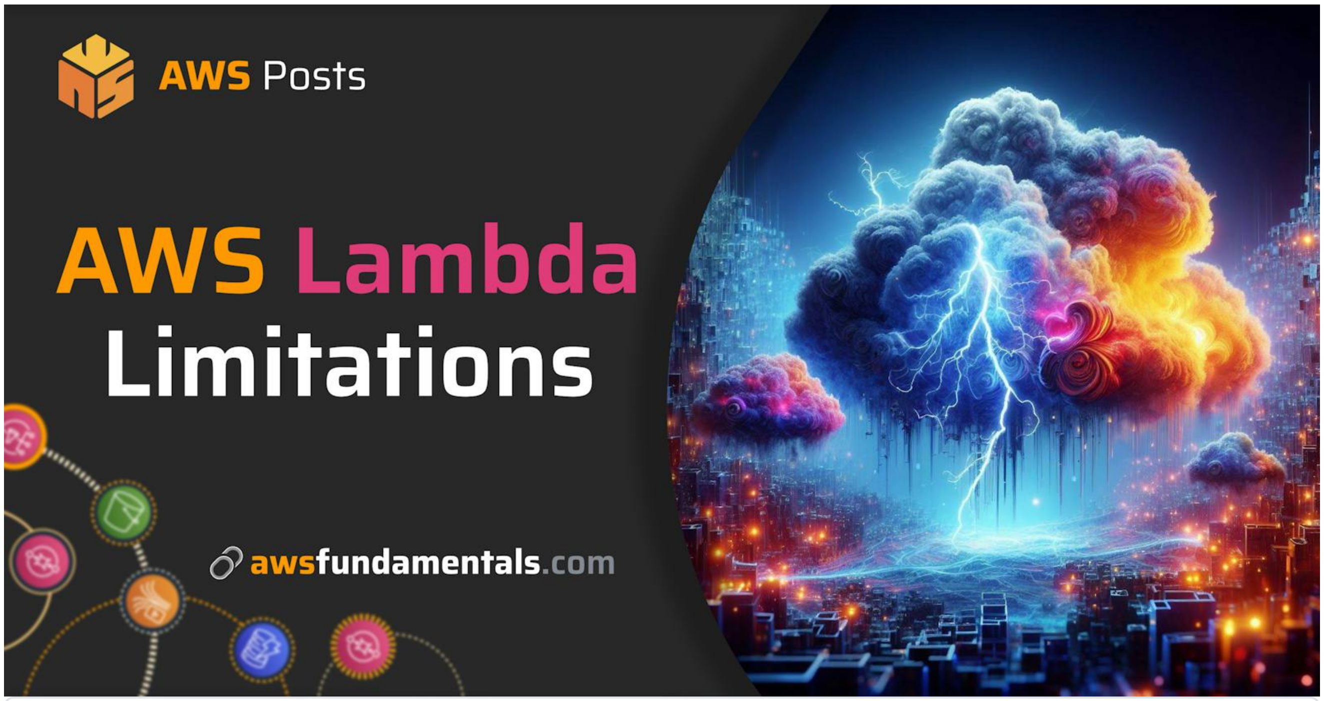
The Hard and Soft Limits of AWS Lambda and How to Mitigate Them.

blog.awsfundamentals.com/lambda-limitations

Tobias Schmidt

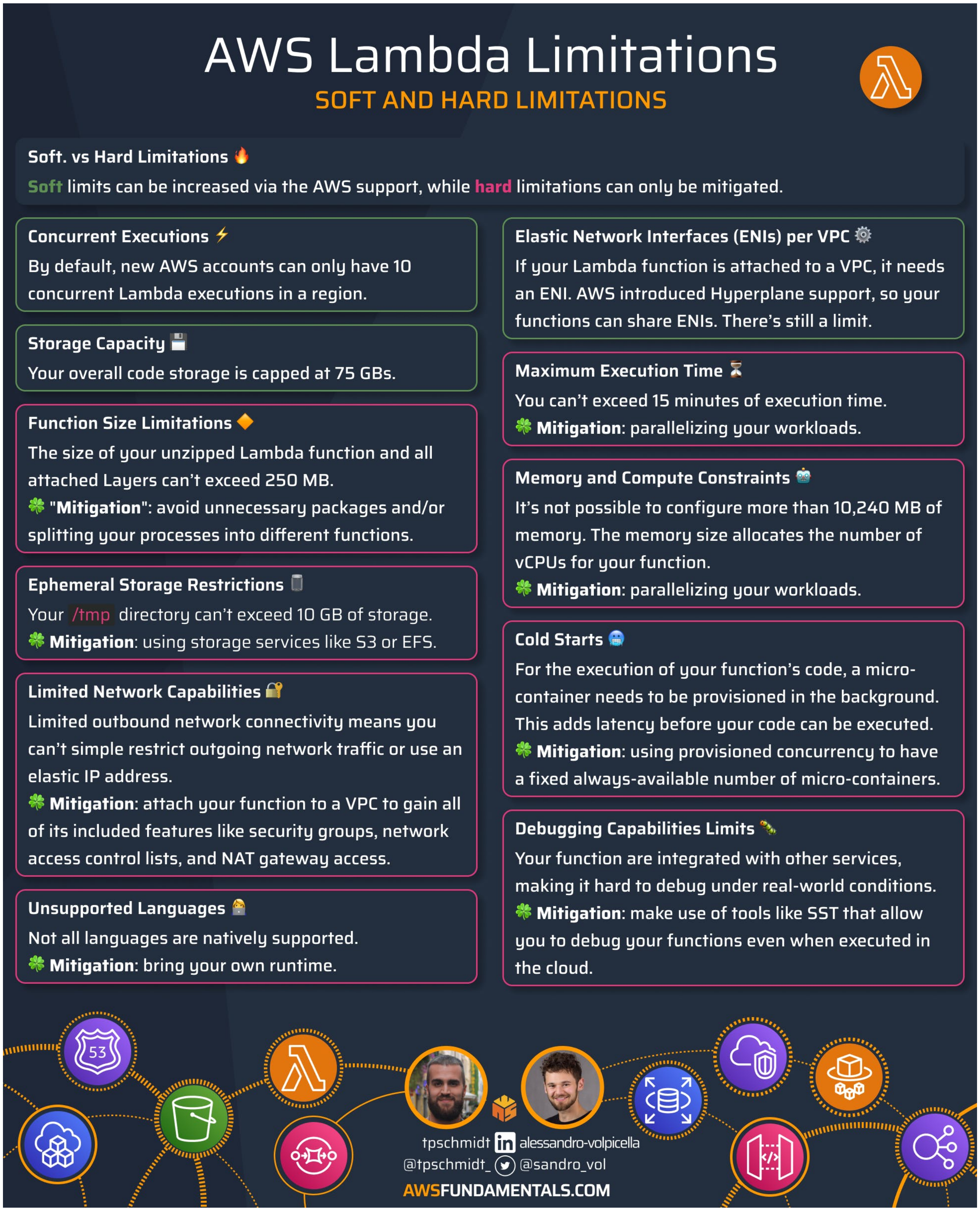


Play this article

0:00 / 12:29

AWS Lambda is the key player in Serverless architectures on AWS. It allows you to build complex systems without worrying about the underlying infrastructure.

However, despite its many advantages, Lambda has some limitations that a developer should be aware of before choosing it for their next project.



AWS Lambda Limitations

SOFT AND HARD LIMITATIONS

Soft. vs Hard Limitations 🔥

Soft limits can be increased via the AWS support, while **hard** limitations can only be mitigated.

Concurrent Executions ⚡

By default, new AWS accounts can only have 10 concurrent Lambda executions in a region.

Storage Capacity 💾

Your overall code storage is capped at 75 GBs.

Function Size Limitations 🔶

The size of your unzipped Lambda function and all attached Layers can't exceed 250 MB.

🍀 "**Mitigation**": avoid unnecessary packages and/or splitting your processes into different functions.

Ephemeral Storage Restrictions 🗄️

Your `/tmp` directory can't exceed 10 GB of storage.

🍀 **Mitigation**: using storage services like S3 or EFS.

Limited Network Capabilities 🔐

Limited outbound network connectivity means you can't simple restrict outgoing network traffic or use an elastic IP address.

🍀 **Mitigation**: attach your function to a VPC to gain all of its included features like security groups, network access control lists, and NAT gateway access.

Unsupported Languages 🙈

Not all languages are natively supported.

🍀 **Mitigation**: bring your own runtime.

Elastic Network Interfaces (ENIs) per VPC ⚙️

If your Lambda function is attached to a VPC, it needs an ENI. AWS introduced Hyperplane support, so your functions can share ENIs. There's still a limit.

Maximum Execution Time ⏳

You can't exceed 15 minutes of execution time.

🍀 **Mitigation**: parallelizing your workloads.

Memory and Compute Constraints 🤖

It's not possible to configure more than 10,240 MB of memory. The memory size allocates the number of vCPUs for your function.

🍀 **Mitigation**: parallelizing your workloads.

Cold Starts 🥶

For the execution of your function's code, a micro-container needs to be provisioned in the background. This adds latency before your code can be executed.

🍀 **Mitigation**: using provisioned concurrency to have a fixed always-available number of micro-containers.

Debugging Capabilities Limits 🐛

Your function are integrated with other services, making it hard to debug under real-world conditions.

🍀 **Mitigation**: make use of tools like SST that allow you to debug your functions even when executed in the cloud.

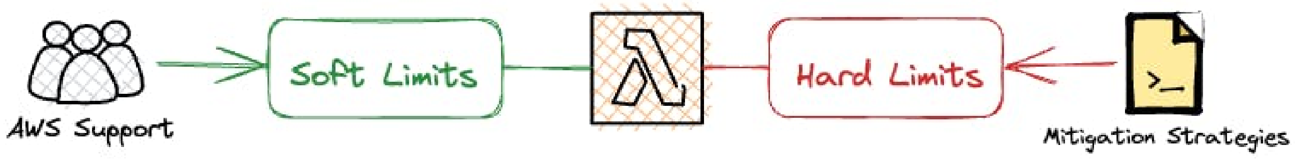
tpschmidt alessandro-volpicella

@tpschmidt_ @sandro_vol

AWSFUNDAMENTALS.COM

Purpose of This Article

This article will cover Lambda's hard and soft limitations, including execution time, memory and vCPU configuration, temporary storage, cold starts, function size, unsupported languages, limited network access, and debugging capabilities.



Soft limitations can be increased via the AWS support, while hard limits act as a fixed set of boundaries. For the hard limitations, this article provides common workarounds and solutions to mitigate those to a certain extent.

Soft Limits

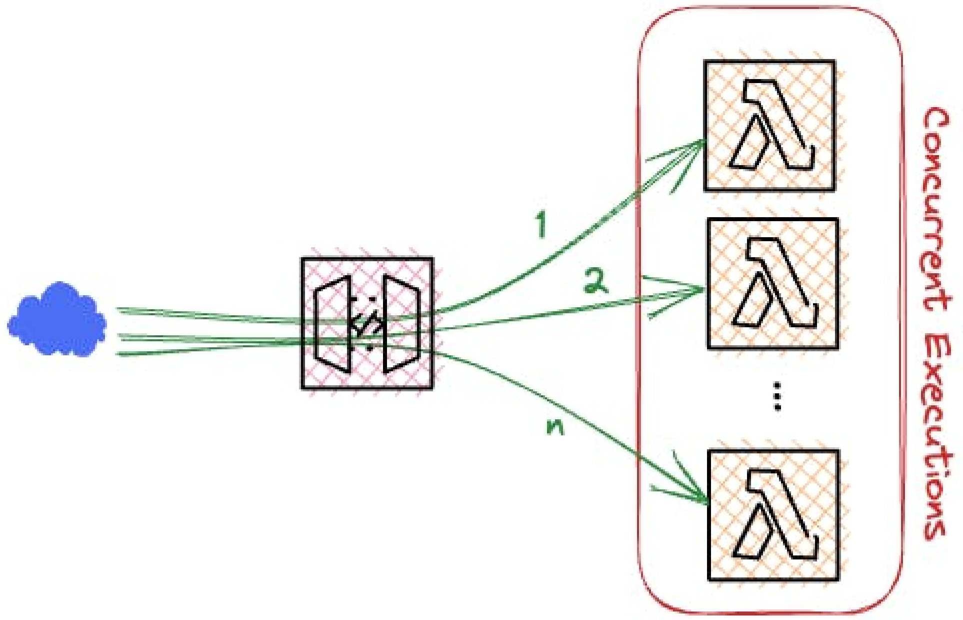
The next paragraphs will elaborate on the current soft limitations of AWS Lambda. Those limits can be increased by AWS support.

A quick overview of Lambda's soft limits:

- The number of parallel executions of your functions per region.
- The storage capacity for the code of your functions and layers.
- The number of elastic network interfaces attached to functions.

Concurrent Executions

If a Lambda function is invoked, AWS will start a micro-container in the background to execute the function code. After finishing the request, this container will be kept alive for a specific amount of time so it can take on the next incoming request.



As each of those micro-containers can only work on a single request at a time, a second concurrent request will need another container. AWS puts a limit on how many concurrent micro-containers can be active at the same time. This means there is a limit on how many parallel executions are possible for your function.

In the past, this limit was set to 1,000 parallel executions per region. AWS drastically reduced this limit into the low double-digits.

You can quickly increase the maximum concurrent execution limit via the support.

Storage Capacity for Function and Layer Code

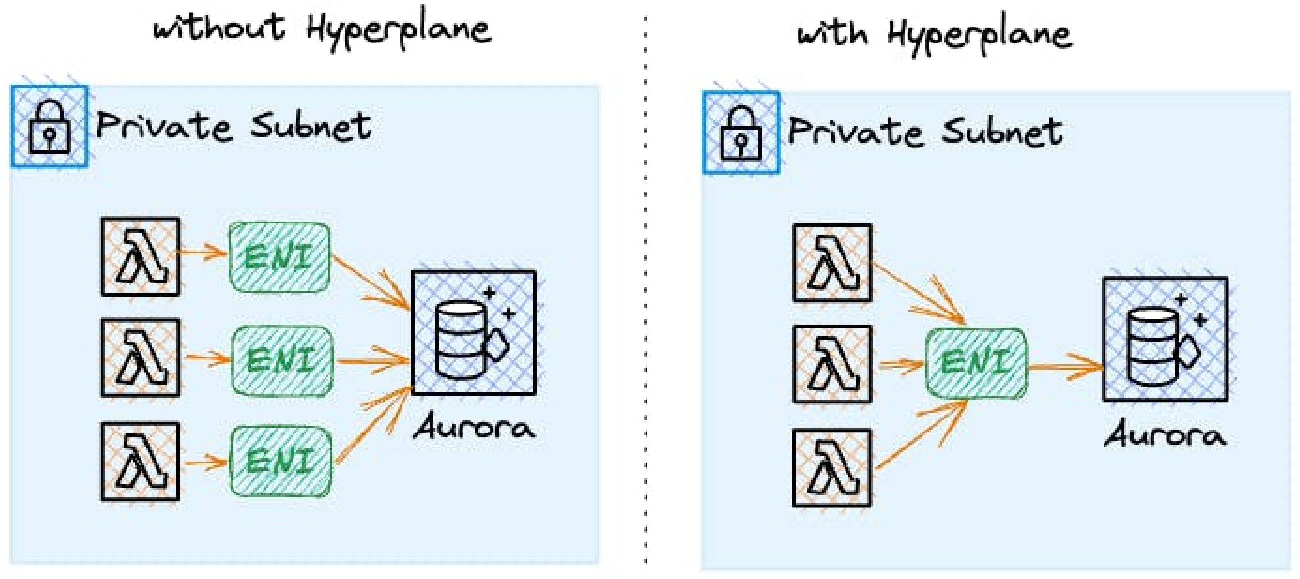
Your function's code and the code of your Lambda Layers need to be stored at AWS. There is a limit to the amount of storage that can be used. Currently, it's at 75 GB.

If you need more storage, this can be increased via AWS support.

Elastic Network Interfaces per VPC

If your function is attached to a VPC, your function's execution needs an elastic network interface (ENI) to communicate with other resources in the network and the outbound internet.

Even though AWS heavily reduced the required number of ENIs for multiple Lambda micro-containers with the introduction of AWS Hyperplane, there's still an upper limit you could theoretically hit. Even though this is very unlikely.



Lambda in the same subnet can reuse the same Hyperplane ENI with up to 65,000 connections/ports. Before the Hyperplane introduction back in 2019, each Lambda micro-container needed its own ENI. This meant two things:

- cold starts time are drastically increased as the creation & attachment of ENIs are very time-consuming.
- the number of available IPs in the subnet can be consumed very quickly, making it impossible to launch new resources or start more function containers.

Hard Limits

Besides soft limits, Lambda comes with a set of hard limits that can't just easily be improved by the support.

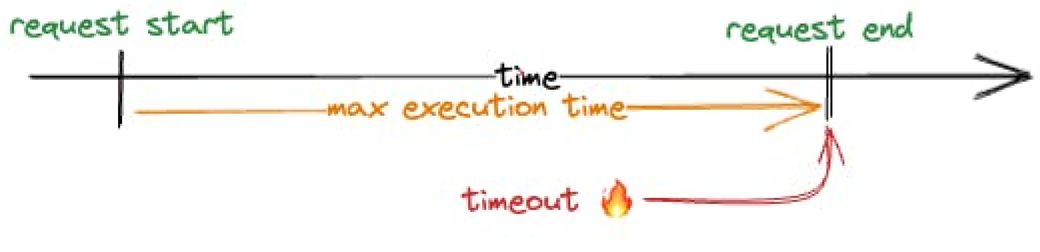
Nevertheless, there are strategies to get better results and somehow mitigate the limitations.

Here's a list of what can't be increased over a certain limit or general implications that you have to deal with:

- The maximum duration of the execution of a single function.
- The maximum memory in megabytes (and therefore vCPUs) for a function.
- The maximum ephemeral storage capacity in the temporary directory of the Lambda environment.
- Cold starts - the time that's needed to bootstrap your function's execution environment.
- The maximum size of your function's code and all attached layers.
- The natively supported languages.
- The network capabilities for AWS Lambda.
- The debugging options.

Maximum Execution Time

AWS Lambda has a configurable maximum execution time limit of up to 15 minutes. If this limit is reached, the function will be stopped forcefully by AWS. This means that long-running processes are not easily possible with Lambda.

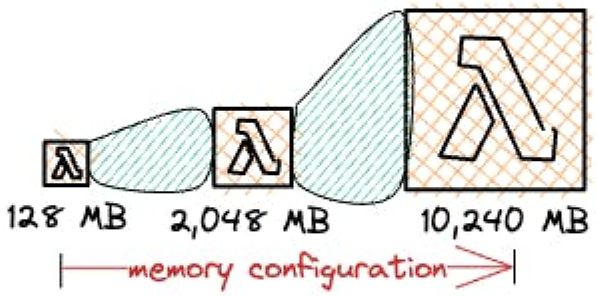


One common approach to mitigate this issue is to break down long-running processes into smaller tasks. Due to Lambda's great support for parallel workloads, this doesn't have to be a show-stopper but can result in faster processing times.

- 🔴 **Limitation**: 15 minutes of execution for a single function.
- 🍀 **Mitigation**: parallelizing your workloads.

Memory and Compute Constraints

The maximum memory you can configure for a Lambda function is 10 GB. The memory size also correlates with the vCPUs that are available to your function, meaning higher memory configuration will result in faster computation times.



This can be mitigated by optimizing your code to use less memory or by splitting memory-intensive tasks into smaller sub-tasks. Remember that AWS Lambda is a great choice to run operations with high concurrency as each Lambda micro-container will receive the assigned compute capacities.

- 🔴 **Limitation**: 10,240 MB of memory.
- 🍀 **Mitigation**: parallelizing your workloads.

Ephemeral Storage Restrictions

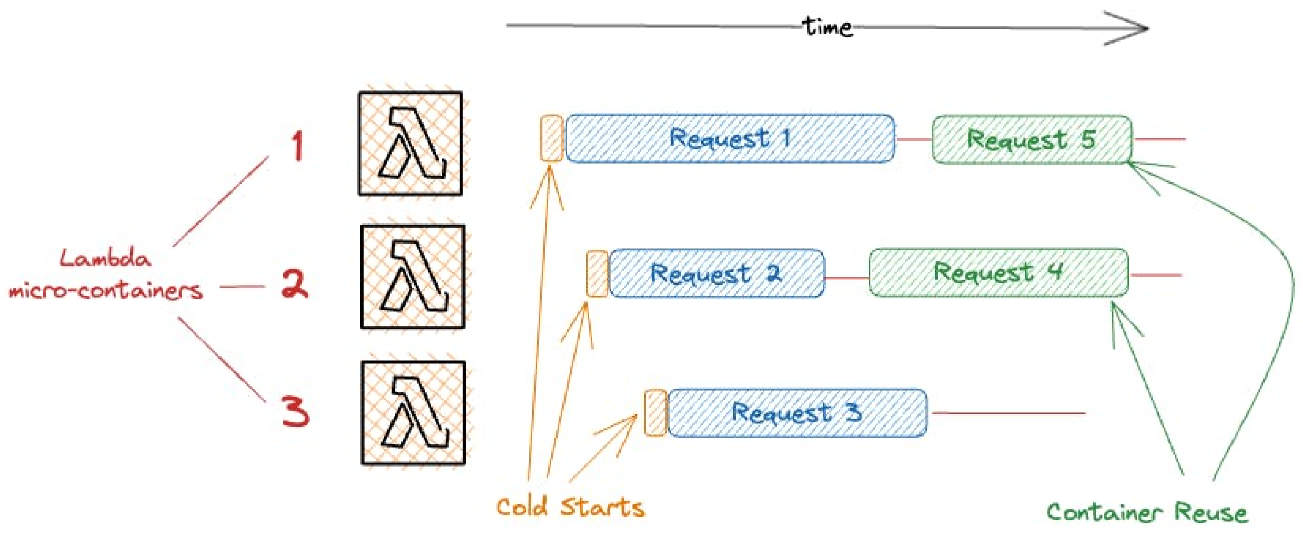
Lambdas have limited temporary storage capacity for the ephemeral directory `/tmp`. You can increase the default size of 512 MB up to 10 GB. Those upgrades are not free but will increase how much you're charged per GB-second.

A common approach to mitigate this is to use services like Amazon S3 or Amazon EFS for storing temporary data. Don't forget that this will also introduce additional costs. EFS also comes with the downside that your Lambda function needs to be attached to a VPC. This will then increase cold start times.

- 🔴 **Limitation**: 10,240 MB of ephemeral storage.
- 🍀 **Mitigation**: using external storage services like S3 or EFS.

Cold Starts

Serverless doesn't mean that there are no servers - they are just not your responsibility. Each of your requests will be handled by a micro-container that is started and managed by AWS and will run for a certain time. After that resources are freed again.



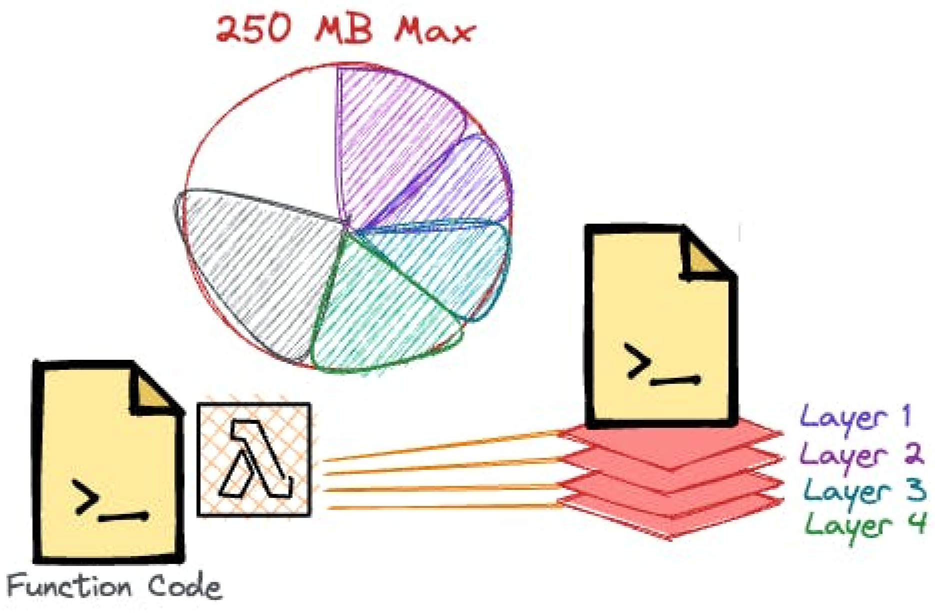
The bootstrapping process of such a micro-container is called cold start. Each cold start will add additional latency to your request. As each micro-container can only handle one request at a time, concurrent requests will need to start multiple micro-containers.

You can mitigate this issue by using Provisioned Concurrency. Provisioned concurrency will keep a configured number of these micro-containers warm at all times, meaning that there's no cold start. As with other mitigations, this will introduce additional costs as provisioned concurrency doesn't come for free. It will also slow down your function deployments from a few seconds to a few minutes.

- 🔴 **Limitation**: AWS regularly de-provisions the execution environment of your function to free resources.
- 🍀 **Mitigation**: using provisioned concurrency to have a fixed always-available number of micro-containers.

Function Size Limitations

That's maybe one of the most critical limitations: the unzipped code of your Lambda function can't exceed 250 MB. This includes all the attached layers.



There's no easy mitigation except for keeping an eye on your dependencies. Think carefully if you need another dependency, as mostly not your code will be the problem. Besides that, you can try to split one large function into multiple smaller ones. Don't forget the trade-off that more functions will always result in more cold starts.

- 🔴 **Limitation**: your extracted function and layer code can't exceed 250 MB.
- 🍀 **"Mitigation"**: keep an eye on your dependencies and avoid unnecessary packages and/or splitting your processes into different functions.

Unsupported Languages

Natively, AWS Lambda supports a limited number of programming languages. They include the most popular ones like Node.js, Python, Go, and Java though.

Luckily, AWS Lambda supports custom runtimes. This means you're able to provide any runtime and therefore use any language possible.

- 🔴 **Limitation**: AWS only offers native support for a subset of prominent programming languages.
- 🍀 **Mitigation**: bring your own runtime.

Limited Networking Capabilities

Lambda has limited outbound network connectivity. For example, you can't get a static IP address which is often needed for security-critical third-party APIs in enterprise applications. Furthermore, you can't easily set up security rules that will restrict outgoing traffic.

This can be mitigated by attaching your Lambda function to a VPC and using a NAT Gateway for outbound calls.

- 🔴 **Limitation**: AWS Lambda doesn't come with advanced options to observe or restrict network traffic of your functions.
- 🍀 **Mitigation**: attach your function to a virtual private network to gain all of its included features like security groups, network access control lists, and NAT gateways.

Debugging Capabilities Limits

Debugging with Lambda is tricky, as you can't simply run your function locally as it's always integrated with other native AWS services.

One mitigation for example is to use the SST framework which allows you to debug your Lambda functions on your local machine. This includes setting breakpoints, as SST can directly connect to your Lambda function via a web socket connection and intercept the actual invocation.

- 🔴 **Limitation**: an AWS Lambda function that's integrated with a lot of other AWS services is hard to debug.
- 🍀 **Mitigation**: make use of external tools like SST that allow you to debug your functions even when executed in the cloud.

Lambda Soft and Hard Service Quotas

All of the soft and hard limitations for AWS Lambda can be found on the service quota page. They apply to every region and as mentioned in the article, soft limits can be adapted via the support.

Make sure to frequently visit the service quota page as AWS regularly adjusts the quotas for Lambda.

Conclusion

AWS Lambda is a powerful and flexible service that offers many benefits, including serverless architecture, scalability, and cost-efficiency. However, as with any technology, there are also limitations that developers should be aware of. From execution time and memory limitations to cold starts and unsupported languages, the limitations can pose challenges for developers looking to build and run serverless applications powered by Lambda.

Fortunately, there are various workarounds and solutions available to mitigate some of these limitations.

As AWS continues to evolve and expand its services, many of the current limitations of Lambda will likely be addressed and new features and capabilities will be added to further enhance its functionality.

Frequently Asked Questions

1. **What is AWS Lambda?**
   AWS Lambda is a serverless computing service provided by Amazon Web Services (AWS). It allows you to run your code without provisioning or managing servers.

2. **What are the limitations of AWS Lambda?**
   AWS Lambda has both soft and hard limitations. Soft limitations include limits on concurrent executions, storage capacity for function and layer code, and elastic network interfaces per VPC. Hard limitations include maximum execution time, memory and compute constraints, ephemeral storage restrictions, cold starts, function size limitations, unsupported languages, limited networking capabilities, and debugging capabilities limits.

3. **Can soft limitations be increased?**
   Yes, soft limitations can be increased via AWS support.

4. **What are common strategies to mitigate the hard limitations of AWS Lambda?**
   There are various strategies to mitigate the hard limitations of AWS Lambda. For example, you can break down long-running processes into smaller tasks to mitigate the maximum execution time limit. You can optimize your code to use less memory or split memory-intensive tasks into smaller sub-tasks to mitigate memory and compute constraints. You can use services like Amazon S3 or Amazon EFS for storing temporary data to mitigate ephemeral storage restrictions. You can use Provisioned Concurrency to keep a configured number of micro-containers warm at all times to mitigate cold starts. You can split one large function into multiple smaller ones to mitigate function size limitations. You can use custom runtimes to use any language possible to mitigate unsupported languages. You can attach your Lambda function to a VPC and use a NAT Gateway for outbound calls to mitigate limited networking capabilities. You can use the SST framework to debug your Lambda functions on your local machine to mitigate debugging capabilities limits.

Published on