HELECLOUD

How to overcome the maximum timeout limit in AWS Lambda

Vasil Petrov, Cloud Infrastructure Engineer

HELECLOUD

How to overcome the maximum timeout limit in AWS Lambda

Introduction

What is AWS Lambda? A managed serverless service that helps us run our code without needing to provision any infrastructure for it. And as such a managed service it has its own limitations.

As a best practice, we should set the timeout value based on our expected execution time to prevent our Lambda functions from running longer than intended, because they are billed based on execution time in 100ms increments. Avoiding lengthy timeouts for Lambda functions can prevent us from being billed while a Lambda function is simply waiting to timeout, the reasons could be many, for example, creating an indefinite loop by mistake, etc.

Historically, the maximum execution timeout limit was 5 minutes. At the time of writing this article, AWS Lambda functions cannot run continuously for more than 15 minutes. This is a hard limit in the AWS Lambda service. In practice, there are use cases for which those 15 minutes are not enough.

In this article, we will outline a lift and ship solution on how to overcome the timeout limit in AWS Lambda service.

The Problem

One example is when we have a Lambda function that is scanning thousands of CloudFormation stacks and enabling termination protection on some of them based on pre-defined criteria. Moreover, what happens if we want to expand the scope of the Lambda function across multiple AWS accounts?

Let's assume, for the sake of the example above, that the execution time required for the scan in one or more of the accounts takes roughly fifteen minutes. With this single limitation, we cannot follow a unified way of enabling termination protection of the CloudFormation stacks across our environment.

On a high level, the benefits of having a unified approach are:

- Following the centralized model *(server – client)* of architecture
    - **Single place of deployment –** Enforcing security policies, rolling out changes/ patches, etc.
    - **Ease of maintenance –** Adding or removing accounts from the scope of the solution.

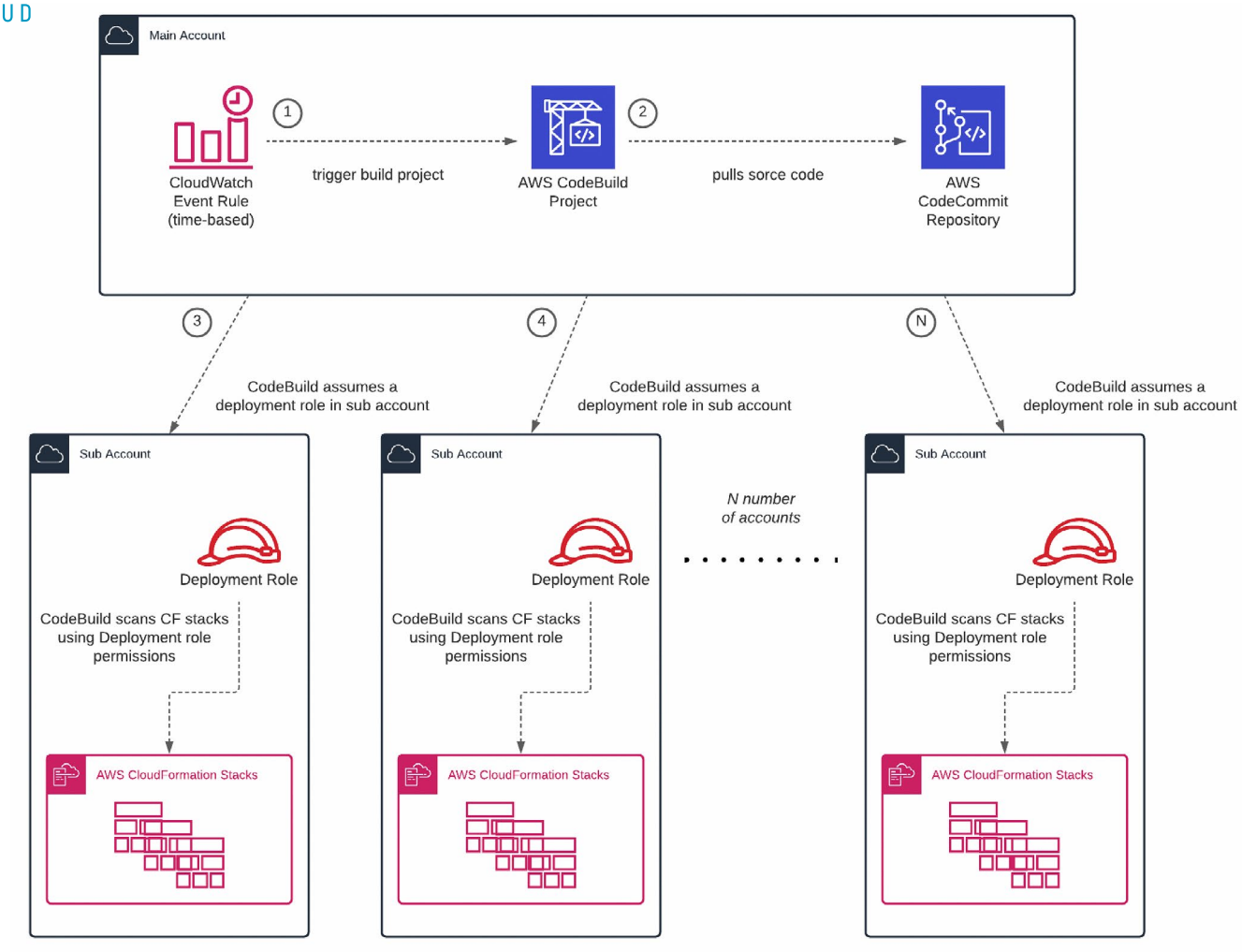
On the other hand, the drawbacks could be:

- Having a complicated solution – Needing to have more experienced people to manage the solution.
- Time consuming – Needing more time for testing possible scenarios before the solution to "go live".

Here I have outlined how to implement a custom solution to overcome the 15th minute timeout limitation in AWS Lambda.

HELECLOUD

The Solution



The core components in our solution are:

- **CloudWatch Event Rule –** There are two types of event rules: time-based and event-based. In our case, we have chosen a time-based one, because we want the CloudFormation stack scan to take place once per day. Event-based rules are more suitable in cases where we want something to happen when a specific event occurs, for example, a Lambda function to be invoked on every merge to master for a specific repository in AWS CodeCommit.
- **CodeBuild –** The main purpose of CodeBuild is to compile source code, run unit tests, and produce artifacts that are ready to be deployed. But in our case, we use it for a slightly different purpose as a run environment for our deployment scripts. Why? – because CodeBuild has a timeout value of up to 8 hours. The default timeout value is 1 hour, but we can adjust it from 5 minutes to 8 hours.
- **CodeCommit –** This is an AWS representation of a source control system. There we store our deployment scripts and keep track of the version control changes. Since we use only AWS services in our solution, CodeCommit integrates easier and faster with AWS services if we compare it with other well-known source control services like GitHub, GitLab, BitBucket, etc.
- **IAM Roles –** Since we have multi-account architecture, IAM roles are playing a crucial role in providing cross-account access. In our case, CodeBuild assumes separate deployment roles per account in order to have the necessary permissions to perform a scan on the CloudFormation stacks and enable termination protection where needed.

HELECLOUD

- **CloudFormation stacks –** A CloudFormation stack is a collection of AWS resources that can be managed as a single unit. In other words, we can create, update, or delete a collection of resources by creating, updating, or deleting stacks.

At this point, we assume the only unanswered question is *"How we have ported a Lambda code into CodeBuild?"* – In our case it is quite straightforward, in our Lambda code we use a simple python code, not using additional custom libraries archived as a Lambda Layer. In CodeBuild we leverage one of the built-in python runtimes.

As a further improvement in our deployment scripts, we can take advantage of multi-threading in Python and put each account in a separate thread. In other words, this would mean that the scan of all CloudFormation stacks would take as much as it would take to scan the account that contains the highest number of CloudFormation stacks. For more information about what is multithreading, how to achieve it in Python, and how to implement it when using Boto3 (AWS SDK for Python), please check *Python – Multithreaded Programming* and *Boto3 – Multithreading or multiprocessing with resources* under Sources section.

Keep the following in mind if you choose to implement multithreading in CodeBuild since CodeBuild has no visibility in with what exit code each thread exists, we can end up having one or more threads failing but CodeBuild to exit successfully. In other words, that means the CloudFormation stack scan failing in one or more accounts, for some reason. In order to avoid this, we need to capture each sub-thread exception and make it visible to the main thread. For more information about how to capture sub-thread exceptions, please check *Handling a thread's exception in the caller thread in Python* under the Sources section.

You need to take into account the fact that, CodeBuild will fail only if it receives an exit code different than "0". Unfortunately, during the time of writing this article, this is not well documented in the AWS documentation. For more information about how to handle popular CodeBuild issues, please check *Troubleshooting AWS CodeBuild* in the Sources section.

There are other options to overcome the limit. One of them would be to decouple the application layer. The architectural options vary a lot here, depending on the level of scale that we are seeking. The AWS services that we would take advantage of would be SQS (Simple Queue Service), SNS (Simple Notification Service), etc.

Another possibility would be to keep everything in the Lambda function. That would mean incorporating logic that would represent the SQS functionality, keeping a state of when the Lambda function timeouts, and on the next run the Lambda function would start from that point. This option has a drawback of a Lambda function becoming more and more complicated in time and managing such Lambda functions might become a nightmare, at some point.

As a summary, in cases when we have simple Lambda functions, and we would like only the expand the scope of the Lambda function, going for a lift and ship solution, taking advantage of CodeBuild for example, would be more suitable.

For cases, when we have more complex Lambda functions, and we would like to expand the scope and the features of the Lambda functions, choosing to decouple the application layer would be more suitable.

HELECLOUD

Sources

AWS increases initial maximum timeout limit:

https://aws.amazon.com/about-aws/whats-new/2018/10/aws-lambda-supports-functions-that-can-run-up-to-15-minutes/

AWS Lambda maximum timeout limit:

https://docs.aws.amazon.com/whitepapers/latest/serverless-architectures-lambda/timeout.html

AWS CloudWatch Events:

https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/WhatIsCloudWatchEvents.html

AWS CodeBuild:

https://docs.aws.amazon.com/codebuild/latest/userguide/welcome.html

AWS CodeBuild Quotas:

https://docs.aws.amazon.com/codebuild/latest/userguide/limits.html

AWS CodeCommit:

https://www.amazonaws.cn/en/codecommit/

AWS CloudFormation Stacks:

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html

Python – Multithreaded Programming:

https://www.tutorialspoint.com/python/python_multithreading.htm

Boto3, an AWS SDK for Python:

https://boto3.amazonaws.com/v1/documentation/api/latest/index.html

Boto3 – Multithreading or multiprocessing with resources:

https://boto3.amazonaws.com/v1/documentation/api/latest/guide/resources.html

Handling a thread's exception in the caller thread in Python:

https://www.geeksforgeeks.org/handling-a-threads-exception-in-the-caller-thread-in-python/

Troubleshooting AWS CodeBuild:

https://docs.aws.amazon.com/codebuild/latest/userguide/troubleshooting.html

HELECLOUD

About HeleCloud

HeleCloud™ is an AWS Premier Consulting Partner based in United Kingdom, The Netherlands, and Bulgaria. HeleCloud™ provides strategic technology consultancy, engineering, and 24/7 Cloud-based managed services. By taking advantage of everything the Cloud has to offer, organisations can reduce operational costs, accelerate innovation, focus on their core business, and have more confidence around data security and compliance.

Contact us

London, UK
6 Snow Hill, London
EC1A 2AY, UK
T: +44 2081 331 119
E: office@helecloud.com

The Hague, Netherlands
WTC The Hague
Prinses Beatrixlaan 582
2595 BM Den Haag
T: +31 70 240 0021
E: office@helecloud.com

Sofia, Bulgaria
141 Tsarigradsko Shose Blvd
VIP Security Building, Floor 2
1784 Sofia
E: office@helecloud.com

Follow us on