Clean up multiple AWS accounts: automatic resources deletion (with IoT button)

18 August 2023 - 8 min. read

AWS CDK *AWS Organizations* *Landing Zone*

More and more companies are choosing the **Landing Zone approach** to their AWS Accounts management. As a result, our work sometimes requires us to work on dozens or, less often, hundreds of AWS accounts simultaneously for a single project.

Not sure what a Landing Zone is? Check out our series on it!

The most recent example was creating a testing environment in our accounts where we had to perform test deployments before deploying to the customer organization. The organization's size justified this need: almost **400 AWS accounts**, so we could not make any mistakes. To mimic their environment, we created an ad-hoc AWS Organization with a dozen accounts with carefully selected resources deployed.

After the work was done, we had to close those accounts, but **closing multiple AWS accounts full of resources** can be time-consuming. Of course, it must be automated as much as possible!

Closing an AWS Account

Closing an account may appear as easy as pressing the "Close" button, but this action has a variety of caveats. Some depend on the account being a **standalone** account or **a member of an AWS Organization**; others are common for both account types.

Here is a little cheat sheet of the most important points of attention:

- The root user's email address can't be reused after closing the account.
- Domains registered with Amazon Route 53 are not deleted automatically, they should be transferred to another registrar or AWS account, or you can disable the automatic renewal and let them expire.
- Suppose an AWS account reopens during the post-closure period. In that case, there may be charges for the cost of any not stopped AWS services or not deleted resources before the account closing (see "Automatic resources deletion" paragraph).
- After an AWS account closes, any access requests to the closed account's AWS services from other AWS accounts will fail.
- AWS doesn't delete Amazon Virtual Private Cloud (Amazon VPC) peering connections when an account participating in the VPC peering connection is closed.

If the account is a member of an AWS Organization, there are even more points of attention:

- The closed account isn't removed from the organization until the post-closure period has passed.
- Only 10% of member accounts can be closed within a rolling 30-day period.
- In the case of using AWS Control Tower, they must be unmanaged before attempting to close the accounts.

So, automating the process of closing an AWS Account is not an easy task.

If the accounts are standalone, it probably means they are few in number, so closing them manually is the best choice; it’s not even worth the time to try to automate it!

Instead, in case they are part of an organization, that's a different story, but, in this case, it won't be possible to close more than 10% of them per month. Not so efficient…

The most efficient solution to this problem is probably to reuse the accounts, perhaps changing their alias, for another purpose instead of closing them. These accounts can be cleaned up, placed in a specific organizational unit (OU), and used for development or testing.

Automatic resources deletion

Another common caveat between both account types is

"When you close your AWS account, you must terminate all your resources, or you might continue to incur charges.".

That's frightening!

Let's explain this part further: when an AWS account is closed, it's not *really* closed; it's in the *post-closure period*, a window of 90 days after closing the account. In this period, a user can still sign in to the account, view past billing, pay for AWS bills, and… incur charges.

When an AWS account is closed, the on-demand billing for the resources stops, but some other cost sources are not stopped (e.g., AWS Marketplace subscriptions). So **it's a best practice to delete all resources before closing the account**.

AWS suggested path

AWS recommends two ways of checking for active resources:

- Through the **Billing dashboard**: check under the **Bills** section.
- Use the **Tag Editor** inside the **Resource Groups** console page by selecting "*All Regions*" and "*All Supported Resource Types*".

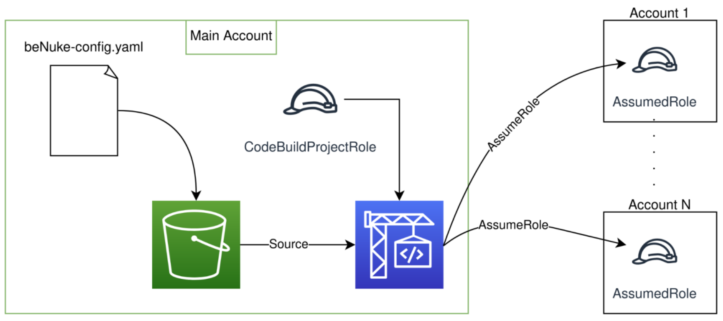
These are both valid solutions for finding active resources, but no solution is offered for terminating them automatically.

beSharp-nuke

To solve this issue, we decided to implement a solution as minimal as possible to delete all the resources in one or more AWS accounts.

Of course, most of the work was already done by aws-nuke, developed by rebuy; it is an awesome open-source tool that can *nuke* (as in "delete all resources") an AWS account. Using a YAML config file it is possible to filter some resources to spare or create a blocklist of production accounts that should never be touched. Unfortunately, though, it's designed to be run on a single AWS account, but we needed a cross-account solution!

The final solution, summarized in the diagram below, is fairly minimal: a CDK template containing just an Amazon S3 Bucket to store the configuration and CodeBuild as the compute part. During the "build" phase, it will parse the configuration searching for the target accounts and run aws-nuke sequentially in every account.



Now, why AWS CodeBuild for the compute part? It was preferred for its minimal needs in resources and its managed nature. Here is a summary of the rejected options:

- AWS Lambda: the 15-minute timeout is not suitable for deleting big accounts.
- Amazon EC2: requires time to start/wake up. It also needs a VPC; the solution must work with minimal requirements and not depend on existing resources.
- Amazon ECS: requires a VPC too.
- AWS Step Function: does not solve the 15-minute timeout if using Lambda because the aws-nuke execution can not be split into multiple Functions. If used just as an orchestrator is probably overkill.
- AWS Batch: seems the best choice for long tasks such as nuking an entire organization, but it requires a VPC.

Usage

All the code is available on GitHub.

First of all, some warnings:

- It's **VERY IMPORTANT** to filter the AWS Role/User used to access the AWS Account. A standalone account can still be accessed using the *root account*, but accounts that are part of an AWS Organization may not have it, so you may be locked out. A filter is already in place in the sample configuration, but make sure to always check and understand before deploying.

- aws-nuke can bypass *Termination Protection*. The sample configuration is designed to do this, so be sure to look for the “disable-deletion-protection” flag and edit it to suit your needs.

Now that you are aware of the major risks involved in this activity, the first step is to decide which accounts to target. In each of them, the deployment of `assumed-role.yaml` template is required, this is an AWS CloudFormation template, so the deployment through **StackSet** is trivial.

Once this prerequisite is deployed, it is sufficient to edit the configuration files and deploy the CDK template inside an account of choice.

There are two configuration files:

- `bucket_content/beNuke-config.yaml` containing the aws-nuke configuration.
- `parameters.ts` contains the Account ID and Region where the solution is deployed and the name of the resources.

When everything is deployed, it is just a matter of running the CodeBuild manually, by CLI, on a schedule using EventBridge, or in any other way...like with an AWS IoT Button!

Bonus: AWS IoT Button as a trigger

Imagine the sense of power derived by nuking an AWS Account and pressing a button. It is possible to trigger the CodeBuild job using a Lambda Function triggered by an IoT Button. To deploy the Lambda is sufficient to set “b*uttonEnabled = true”* inside the template parameters. Unfortunately, a manual step on the AWS Console is required to link the IoT Button as a trigger for the Function.

More detailed instructions are available inside the README.md file in the source code.

Further development

A cool addition to this solution would be a report of the nuked resources, including a list of target accounts, failed resources, and other information. That’s not an easy task though; the output of aws-nuke is not suited to be put directly into a report, it’s saturated with information, most of it not really useful to the end user. The output should be parsed for the relevant information, and a short summary could be sent to a specified email address using Amazon SNS.

Anyway, the output of aws-nuke could definitely be improved, a lot of errors are logged in the standard output. We are currently working on some improvements, which we will submit as pull requests to their repository.

Conclusions

Closing an AWS Account is not easy as it sounds, but paying attention to a couple of recommendations from the documentation is not that difficult either.

Unfortunately, deleting all the resources is a different matter; proceeding manually, especially in cases where the resources span multiple accounts, is prohibitive. But thanks to the right tool and a little CDK code can be done in a lot less time and become a funny, albeit very stressful, task.

We hope this DevLife-changing cheat will be helpful. Let us know in the comments!

See you again in 14 days with a new article on Proud2beCloud!

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!

Andrea Pusineri

DevOps Engineer @ beSharp. I love solving problems and I'm back belt of finding them. Linux enthusiast and security guy wannabe, I like to play CTFs, but in my spare time I'm an

avid comic/manga/book reader. btw I use Arch
