Building iOS Apps with AWS CodeBuild: Pros, Cons, and Our Alternative Solution

23 October 2024 - 7 min. read

AWS CodeBuild | *AWS CodeBuild MacOS Builds* | *CI/CD*

Introduction

Have you ever had to create an iOS application on AWS?

If the answer is no, then this article is for you. And if the answer is yes, this article is still for you!

Recently, we were tasked with developing an iOS application on AWS for one of our clients. With no alternatives available at the start of the project, we designed a custom solution, which we initially planned to share in this article.

However, the AWS ecosystem is constantly evolving, and, as luck would have it, AWS CodeBuild now supports Mac Builds.

Why not explore this new possibility right away? What advantages does it bring? How does it differ from the solution we developed?

We incorporated these considerations into an article, so here we are.

Let's dive into the topic without further ado, starting with AWS's solution.

'Keep in mind that most of the issues mentioned below are due to the current immaturity of the service. It is new and still brings some limitations that AWS will surely mitigate over time.'

AWS CodeBuild for Apple Systems

With the release of the AWS CodeBuild MacOS image, we now have the ability to integrate a machine with a MacOS operating system directly into a pipeline without needing to create custom integration methods. This greatly simplifies and accelerates the infrastructure release, and this is a significant advantage of this solution.

To keep things organized, let's break everything down into pros and cons below:

PROs

Serverless

AWS CodeBuild is inherently serverless, allowing us always to have a machine available to release our applications without worrying about potential downtime or manually managing high availability. In other situations, this typically requires having a second machine in a different Availability Zone.

Cost management

AWS CodeBuild offers the ability to use the same fleet across multiple projects, enabling cost-sharing across several applications and thus reducing the cost per application. However, careful attention will be needed regarding how to manage the setup and configuration dynamically so the same scripts can be reused for each deployment, only changing variables as needed.

In the long run, this approach provides both economic and management benefits across multiple projects.

CONs

One image available

At this moment, only a single image is available, and it comes with only one specific version of Xcode (the program used to create the .IPA file, which is then released on the App Store).

This leads to several issues, such as:

- Not always having access to the latest version of the software.
- Some essential programs (like IXGuard) are missing.
- Difficulty keeping the local development environment aligned with the cloud environment, forcing developers to use a specific version and missing out on updates

that provide better security and performance.

Costs

Additionally, this specific version of AWS CodeBuild has a different payment model than the classic pay-per-use approach typical of more common images (like Linux). The difference arises from the hardware being a physical Mac machine, which requires creating a CodeBuild Fleet. This fleet incurs costs even when it is not actively in use.

This shouldn't be a major issue since the alternative is using a physical Mac instance. However, the cost of CodeBuild is higher than an EC2 instance, especially when considering long-term savings plans.

Complexity

Personally, I expected the integration of some commonly used scripts for app development into the CodeBuild image. This would have made certain necessary operations, like configuring and installing provisioning profiles (required to release an app on the App Store), much easier. Instead, some must be created manually.

Recently, AWS has provided another tool: direct integration between CodeBuild and GitLab Runners. This could be useful when a GitLab environment exists for Software Version Control, allowing seamless integration between the two providers. This can reduce costs and make it easier to manage the instance via GitLab.

Our custom solution

A few months ago, we needed something to autonomously manage the creation of an iOS application since no alternatives were available. So, we started exploring where and how we could integrate automation.

This led us to the solution that is currently in place.

One of the client's prerequisites was to keep everything within the same AWS environment. This meant we couldn't use GitHub Actions, which would have simplified our integration with the iOS environment, as they are directly integrated with a machine equipped with the iOS environment. Additionally, GitHub Actions have the advantage of being more mature as a technology, with more community support, which results in more examples and pre-built integrations.

That said, we had to start from the assumption that the only way to release an iOS application on AWS using managed services and without relying on third-party tools (like a

Docker MacOS image) was to use an EC2 Mac instance.

We then had to figure out how to integrate this machine into a pipeline, as AWS doesn't provide ready-made integrations. The path we considered best was to use a StepFunction to orchestrate the various operations needed to build and release the application.

The idea was to check whether another environment was already using the machine and, if so, wait for the build to finish to avoid doubling the release time for both.

At that point, the next challenge was communicating directly with the machine through this step function. After assessing the requirements against our options, we concluded that we could query the machine via SSM using the "SendCommand" API.

The commands were necessary to trigger the various.SH scripts required for the machine setup and the application release process.

Let's dive into Pros and Cons:

PROs

Custom

Like all custom-built systems, a solution built from scratch allows us to tailor it to our needs and include what we consider truly useful. In this case, we were able to add additional tools like IXGuard and always maintain the latest version of Xcode, which not only simplifies the developers' work but also aligns with the client’s requirements.

Economic advantages

The cost is lower when comparing a single instance with a single CodeBuild, thanks to the use of many serverless services like Lambda and Step Functions. Additionally, the cost of EC2 can be reduced through Savings Plans.

CONs

Management

Certainly, managing the integration logic with the EC2 instance is much more complex than using an out-of-the-box solution like AWS CodeBuild, provided by AWS. However, having our own machine and custom integration allows for much greater control and flexibility, as mentioned above.

Not high available

The EC2 instance, however, is a single physical instance located in a single Availability Zone (AZ), which goes against high availability principles. If an AZ goes down, our machine will no longer be usable, leading to downtime.

Updates management

Another issue arises when there's a need to update or upgrade the operating system or an application; this can cause service interruptions. After a "major version" update, some functionality might change, leading to unexpected errors that would take time to resolve, as they cannot be predicted before the upgrade.

To address the issue of updates, we found it very useful to have a second instance in a shutdown state, with a pre-configured AMI aligned with the production machine. This instance can be used when necessary to test updates, allowing us to make informed decisions on how to proceed, given that updates can always be scheduled. It's also possible to keep the instance running, but this comes with non-negligible costs, so it depends on what you consider more important.

Conclusion

Is one solution better than the other?

Maybe… but we can't say it yet. The answer, as always, is “it depends”: if you’re looking for a fast and well-integrated solution, relying on AWS CodeBuild might be the right choice.

Instead, if you need more control and flexibility, an EC2 instance with Step Functions and Lambda could be a better fit.

Both solutions, ours and AWS’s, still require creating and managing the necessary scripts to install prerequisites, build the .IPA file, and release it; this is by far the most complex, labor-intensive, and difficult part to manage.

If you’d like some advice or examples, let us know. It would be a perfect “Part II” of this article!

That’s all for now.

See you in the next 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!

Stefano Salvaneschi

DevOps Engineer @ beSharp. I’m the kind of guy that can’t let a question go unanswered and I can’t resist making them too! Passionate about IT since i was a child, now landed on the Cloud side.In my free time i enjoy playing videogames, dungeons & dragons and everything geeky in between, all accompanied by a nice beer!
