Serverless Compared: AWS, GCP, Azure – Evolution & New Feature Proposal

When it comes to serverless computing, the top three services that come to our mind are, AWS Lambda, Azure Functions, and Google Cloud Functions. All of these services have come a long way in terms of reducing costs, introducing new features, and improving speed/latency issues. In terms of languages supported by these vendors, Lambda supports **Node.js, Python, Java, Go, .NET**; Azure Functions supports **Python, Javascript, Java, Powershell, F# and .NET**; Google Cloud Functions supports **Node.js, Python, Go, Java, Ruby, PHP, .NET**. The maximum time for execution differs heavily for these vendors with Lambda only allowing **15 minutes** of execution, Azure allowing **unlimited** execution with premium membership and Google allowing up to **60 minutes** in their Gen 2 functions. These different options also use a unique way to optimize the start time for applications. AWS Lamba, uses **SnapStart** technology (taking a snapshot of an environment and restoring it when a new environment is invoked). Azure functions on the other hand use prewarmed or **always-ready instances**. Meanwhile, Google uses a new technology in their 2nd Gen functions where the instance is built on Cloud Run and allows a single function instance to handle multiple **concurrent requests**.

In the early versions of AWS Lambda (2020 - early 2021), container image support was introduced which allowed you to make docker images locally and then deploy them to lambda which extremely helped incorporating custom lambda layers. Around 2019 to 2021, AWS addressed the cold-start pain with provisioned concurrency that kept initialized execution environments ready to avoid cold starts. It was not until 2022 until Lambda introduced the SnapStart feature (first for Java) which allowed them to sub-second cold starts. In 2024, Lambda expanded managed runtime versions and added ARM64 options for lower per-second cost, increased memory envelopes to make it easier to develop and run complex services. And finally in 2025 they introduced a wider SnapStart support for different runtimes (Python/.NET and etc.).

Even though Lambda has tried minimizing the cold starts, it is still a vital issue as even a 200ms savings in cold starts can improve user experience and rates by a good margin [1]. As a product manager, I would like to incorporate a predictive auto tuning for cold start performance. This means that there will be an AI-driven tuning that will monitor invocation patterns to predict peak periods. Based on that, it will automatically adjust concurrency, SnapStart allocation or runtime hints without human intervention [2]. This solution can also provide metric and cost tradeoffs to the customer if they want to tinker any patterns. In conclusion, this should reduce latency spikes and manual tuning and should cut erratic cold start times with a measurable difference.