How Request Pricing Affects Deployment Architecture

Let's see how request pricing, i.e, charging based on the number of requests, affects deployment architecture.

WE'LL COVER THE FOLLOWING

With serverless applications, developers write functions to coordinate and perform business features unique to their application, and use platform services to manage state or communicate with users. In the case of AWS, the pricing for most of those services is also structured around utilization, not reserved capacity. Amazon Simple Storage Service (S3), a scalable file system, charges users for transferred bytes in and out of the storage service. Amazon Simple Notification Service (SNS), a message topic system, charges for each sent message. The whole platform is designed so that how much you pay depends on how much you use the platform. Lambda is the universal glue that brings all those services together.

Critical aspects of serverless architecture

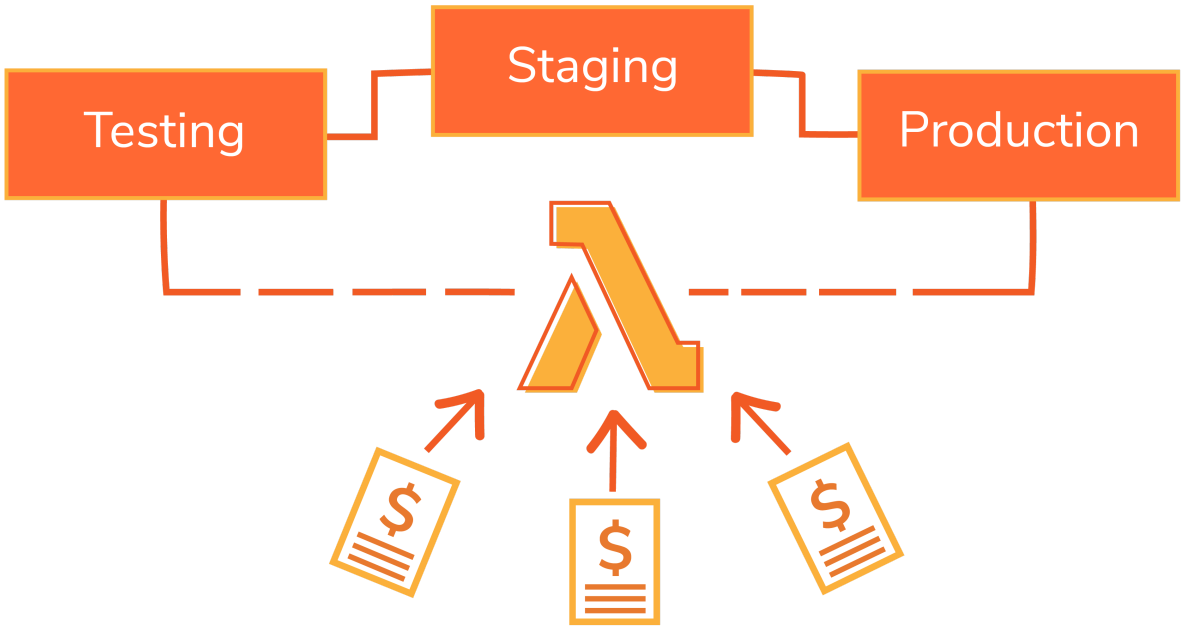
In his conference talk *Why the Fuss about Serverless*, Simon Wardley argued that serverless is effectively platform-as-a-service, or more precisely what platform-as-a-service was supposed to be before marketers took over the buzzword. No doubt history will repeat itself, and in a few years things that have nothing to do with these ideas will be sold as ‘serverless’. But for now, here are what are considered the three critical aspects of a serverless application:

- Infrastructure providers are responsible for handling incoming network requests.
- Operational cost is based on actual utilisation, broken down by individual requests.

- Technical operations (deployment, scaling, security, and monitoring) are included in the price.

These three factors make up an interesting set of deployment constraints. For example, with request-based pricing and no overhead to set up or operate environments, it costs the same amount to send a million user requests to a single version of your application, or two different versions, or 50 different versions.

> ***The number of requests matters, not the number of environments***.



Ever since the time of immortal mainframes, all popular deployment architectures had a significant overhead for creating a new execution environment. This is why staging and testing environments were commonly less powerful than the production environment, and why developers and testers often fought each other for control of a single integration test system in the company. Containers and virtual machines significantly reduced the time it takes to provision a new environment, but they did not change the mathematical formula for operational costs too much. Two equivalent copies of the production environment cost twice as much. That model created strong financial incentives to bundle features into applications that could be deployed together and reuse the same resources.

I recently worked with a consulting client where all the report generators were running on the same cluster, and a single slow report clogged the whole container it would run on. It slowed down work for everything else that

needed to happen. Separating out an environment for each report type would be economically silly since most of them would sit idle most of the time. But when all reports end up in a single cluster, people need to carefully plan capacity and pay for reserved performance or suffer from delays during peak usage. With Lambda, capacity planning becomes Amazon's problem, and there are no specific benefits to bundling different workloads into a single cluster. Instead of aggregating tasks to save money in deployment (which could cause interference and performance bottlenecks), Lambda makes it economically viable to deploy each report type to an isolated infrastructure. The price of Lambda is proportional to the number of requests, not the number of workload clusters, so running a separate set for each report type costs exactly the same as putting all report types in a single combined cluster. Even better, if those report generators have different memory needs, Lambda turns out to be cheaper. If everything is in the same cluster, the containers will need to reserve as much memory as the heaviest report process needs. With Lambda, processes with smaller memory needs can run in smaller containers, and cost significantly less.

> With Lambda, you can create as many different versions of your whole infrastructure as you like without multiplying costs. Instead of developers and testers arguing over a single staging copy, create one for each team. Even better, create a full copy of your application for each developer or tester, with production-like capacity. It costs the same as having a single version because the price is based on the number of executed requests, not the number of environments.

In the next lesson, you'll explore other factors that are affected by request pricing.