The Not-So-Straightforward Road From Microservices to Serverless

Phil Calçado - @pcalcado - philcalcado.com
What are microservices?
Microservices seem to be highly-distributed application architectures.
What is Serverless?
Who knows?
Serverless applications are ones that are implemented using Serverless services. A Serverless service is one that entirely, or very nearly entirely, exhibits **five common traits**. This series of mini articles describes these five traits, namely that a Serverless service:

1. **Requires no management of Server hosts or Server processes**
   (explained below)

2. **Self auto-scales and auto-provisions, based on load**

3. **Offers costs based on precise usage**

4. **Has performance capabilities defined in terms other than host size / count**

5. **Has implicit High Availability**
e.g.
Keep your monolith for as long as possible
Monolith
But what if you waited so long microservices aren’t hype anymore there are other alternatives?
“Chapstick"
This project wasn’t particularly successful
Some challenges we faced
Solving a bug means re-processing all the data
“The write path issue”
Cold start for JVM Lambdas
DynamoDB Costs
Who owns what? What calls what? Is this Lambda supposed to be in production?
Some of the challenges we DID NOT face
Getting new engineers productive on Lambda/DynamoDB
Cold start for Node.js Lambdas
Operations allergy
"Local" development
Developer happiness
How can we keep the good and get rid of the bad?
User Experience

Database

Back-end

User Experience

Database

Back-end

User Experience

Financial Reporting

User Manager

Point of Sale
Serverless Paradox: Abstracting away the runtime is supposed to relieve you from infrastructural concerns and let you focus on the business logic, but instead the pinball machine of lambdas, buckets and queues yields an anemic domain.
We were suffering from Pinball machine Architecture
Public versus Published Interfaces

Martin Fowler

One of the growing trends in software design is separating implementation and public part. The principle here is separating modules and private parts so that the private part will not interact with other modules. However, there is another distinction—the one between published and published interfaces. This distinction is important because it relates to how we work with the interface.

Public versus Published Interfaces

Let's assume I'm writing an application in a more abstract language—to make things concrete, let's assume the language is Java. My application consists of several abstract interfaces, each of which contains a public interface. This...

There's something to be said for the public—published distinction being more important than the more common public—private distinction.
Use Serverless as building blocks for Microservices
AWS Accounts are a great way to define service boundaries.
API Gateway is actually quite expensive (at the moment)
~1/10 of the engineering teams was dedicated to platform
But... is this really Serverless?
Who cares?
You can’t got from 2/10 to 10/10 in one jump.
Serverless looks like the future, but we’re not there yet
Questions?