Kishore Bhatia: What computing model led to serverless?

Nate Taggart: People used to buy physical hardware. Buy a server. Install it in the closet. Somebody’s job was to plug the wires in. If you got more traffic than the server could handle, then buy a bigger server. This is what we called “scaling vertically.” And it worked. It was a very simple model.

But it had some challenges, too. First off, there was no flexibility in scaling. You had to predict in advance how much traffic you were going to have, and buy an appropriately powerful server to handle that. If you hit a limit, you had to make a large capital purchase of a bigger server, which could be [up to] tens of thousands of dollars. As companies got more sophisticated, they started to add multiple servers and then load-balance traffic across them. This gave some flexibility in terms of scaling but still required long-range planning.

About 10 years ago, the cloud started to emerge. Instead of a big capital expense up front, the model was to rent infrastructure at much more granular levels. You could pick different-sized servers and rent them by the hour, instead of committing to them for the lifetime of the server. This model has predominantly been driven by Amazon Web Services [AWS]. For years it’s worked really well. We have been increasing the utilization rate of the servers. Instead of buying a big server that has more capacity than you need, you can now scale up or scale down.

But as applications became more complex, it became harder to manage the underlying infrastructure to coordinate it with application changes. We wanted to improve applications more quickly. We wanted developers to be able to ramp up on a software project more easily. So, we started breaking apart these giant monolithic applications into microservices. Microservices were easier to develop because they were smaller.

But that introduced the problem of orchestration. Orchestration is the idea that individual services can be scaled, turned on, turned off, restarted, and recovered, and can flex up or down
in terms of capacity. And those services can run atop a cluster of compute resources.

However, if an application is now a dozen microservices, you have to individually manage how each of those services scales. Now there are two levels that you can optimize on. You can optimize the underlying infrastructure, or you can optimize at the application level (the individual resources available to each microservice). That works well in terms of managing utilization, and it helps developers ship applications more quickly, but it introduces a large degree of complexity. You have more points where you’re managing health, and that job requires some degree of sophistication.

A large portion of the market—while they have a need for orchestration and they do want to drive efficient use of their infrastructure—don’t see that activity as a business differentiator for their organization. They don’t want to hire and develop a deep orchestration skill set because that doesn’t help their organization meet its goals.

With serverless, they’re able to outsource the need for the orchestration skill set. Let Amazon or Microsoft or Google handle the orchestration layer. Let their team focus on developing applications and managing the health of the applications on top of this new model of managed infrastructure.

Is the trend driving serverless more on the business side? Or is it technology that’s driving this adoption of a computing model that’s more cloud driven and more infrastructure abstraction?

I think it’s fundamentally both. Every business, regardless of the service that they deliver, is looking to ship products more quickly. They want to increase their velocity. They want to be able to create more value and, of course, capture more value from their customers.

At the same time, they want to manage cost. They don’t want to just throw infrastructure at the problem and scale up arbitrarily. They want to be able to ensure that they’re maximizing the utilization of the infrastructure that they’re paying for. And they want to balance that efficiency with the risk that they might go down, or have an error, or in some other way fail to deliver their services.

When we say “serverless,” we do know that there are still servers. How would you define that?

There’s a lot of debate around what is serverless. Some people think of it as a literal compute model. Maybe it is better described as function as a service [FaaS]. This is a code that you develop and will run on demand in the infrastructure provider.

I like to think of it more as a development model where an organization is able to take advantage of managed services and managed infrastructure. The focus may be on the compute level of the application but could, with this definition, also touch on data storage and API delivery as a managed service.

You say it’s a development model. How would you break that into pieces?

Developers have a lot of aspects to their job over the entire software delivery lifecycle. If you focus on a single level, you can potentially increase your productivity dramatically. A developer can add the most [value] by writing code. They may not be particularly skilled at managing the underlying infrastructure. Configuring a network properly is a very different skill set than writing code. There are people who specialize in each. For small organizations it can
be challenging to build a team that has all of the skill sets required to manage the entire lifecycle.

But even for large organizations, if your business can focus on building and delivering products, not on managing boilerplate infrastructure that, frankly, Amazon or Microsoft can do better than you, there's potentially a competitive advantage that you gain. Your business is focused on building and creating value, and your competitors are focused on managing boilerplate hardware.

There's a lot of focus on the FaaS aspect of serverless. Can you define what you mean by functions here?

There are a few emerging patterns for building serverless applications. You can build a lot of applications similarly to how they're built today. Say, for example, you provide an API endpoint and connect it to an FaaS. On every request to that endpoint, your code would run and return a response. That looks very much like a traditional infrastructure model, but there are minor differences: the function is not long-lived; it will start and stop. Because of that there's no state that will carry forward from one transaction to another. But still, it follows the request–response loop that developers are used to.

FaaS also enables other development models. One that is gaining in popularity is the event-driven model. In an event-driven model, you can subscribe your functions to listen for certain event triggers and then respond to them. As an example, you could listen for when a file is uploaded to an object storage bucket. When a file gets uploaded, I want my code to run. That's commonly used for transcoding videos or generating thumbnails. When an image gets uploaded, my function sees that happen, generates a thumbnail, and outputs that to another location.

This event-driven model is interesting because it actually makes applications feel much more distributed. For example, at an enterprise you may have a team focused on marketing that's listening to events that are generated from the e-commerce site. Someone puts a product in their cart and then abandons the cart. You could write a function that looks for when this event happens and then run a function in response to that. Maybe trigger a marketing email that responds to cart abandonment.

Not only is the code distributed on multiple physical servers, but the development model is also distributed. You have different teams relying on and potentially reacting to both code and events that other teams may be responsible for managing.

Are cloud providers offering different models? We covered AWS Lambda, but are there other serverless-infrastructure providers?

There are multiple providers. Amazon is leading the way. Microsoft has a well-developed offering. And Google has an offering, although I believe it's still officially in beta. In all three cases, what we see is that enterprises are not choosing a cloud provider based on the serverless offering. They're taking their serverless compute functions to the cloud that already has their data, their events, their networks, and the authentications they used to deploy to, and that they already have skills developed for.

That said, I also want to step back and talk about the operations point that you mentioned. I think serverless is high in the hype cycle right now. Amazon has shown a lot of growth in their Lambda product. They've featured it at the AWS re:Invent conference in the past couple of years. We've seen increasing investments from Amazon, Microsoft, and Google in their cloud serverless offerings. And in response to that, I think we have gotten a bit swept up by the hype and the buzz.

One of the trends being discussed is this idea of “no ops.” That as a managed service there's no operations responsibility for an enterprise consuming a serverless or an FaaS offering. That is simply not true. There are parts of operations that in fact do get outsourced to the cloud provider. Availability is clearly outsourced. Scaling is outsourced. Orchestration is outsourced.

But there are huge parts of operations that are not being outsourced, such as managing and maintaining the health of the application. And how you build and release an application with automation and reliability—that's not being outsourced, either. If you step back and ask, “Which DevOps responsibilities are we getting rid of, and which ones are we still fundamentally owning?,” you'll find a significant amount of operations responsibility that still resides with the company that's running these applications.

With containers, you can run something stateless for a quick single process and then scale that model on a cluster of available virtual machines. How do you compare that with where serverless is? And you mentioned functions earlier—is that something like containers as a service, which a lot of cloud providers now have? And how does that match up to where serverless is?

Over the last six months, the definition of serverless is starting to look more like a gradient and less binary. I don’t think that there are offerings that are purely serverless and offerings that are absolutely not serverless, with those as the only two categories.

Amazon, for example, recently released a product called Fargate. Fargate
Earlier we mentioned a use case of video transcoding. If you have a 30-second video clip, I think it’s safe to say you could transcode it in 30 seconds. But what if you have a four-hour video clip? You want something that can be longer-lived. Now it may still be transactional. In a case like that, Fargate could be a hybrid solution. It’s not constrained to run in only five minutes. You can run the Docker container as long as needed. But it still looks very serverless. It’s event driven. It’s stateless between transactions. Once again, you don’t have to manage the orchestration, or the scaling, or the availability, or the underlying cluster.

An event can trigger this Docker container to run. With Lambda in particular, you have some constraints on how long a function can run for. The maximum Lambda can run today is 300 seconds.

Earlier we mentioned a use case of video transcoding. If you have a 30-second video clip, I think it’s safe to say you could transcode it in 30 seconds. But what if you have a four-hour video clip? You want something that can be longer-lived. Now it may still be transactional. In a case like that, Fargate could be a hybrid solution. It’s not constrained to run in only five minutes. You can run the Docker container as long as needed. But it still looks very serverless. It’s event driven. It’s stateless between transactions. Once again, you don’t have to manage the orchestration, or the scaling, or the availability, or the underlying cluster.

It’s stateless between transactions. Once again, you don’t have to manage the orchestration, or the scaling, or the availability, or the underlying cluster.

You can also look at some other offerings like managed Kubernetes. Amazon released EKS (Elastic Container Service). Google for a while now has managed Kubernetes. And, of course, there are companies like Platform9 that run real Kubernetes with a managed-service layer.

In each of these offerings you see various levels of outsourcing for the management of the underlying infrastructure. I think Amazon’s model is to give you the big knobs. How do you scale, and what are your scaling rules? What type of infrastructure do you want it to run on? And then they try to abstract the actual running of the Kubernetes cluster as much as possible, or at least simplify that management aspect.

And then there are offerings, of course, where you’re running Kubernetes natively; you’re managing the cluster, and you’re managing all of the orchestration, the service discovery, and the availability and health of the application. That very much does not look serverless. It does not look like a managed service at all.

ABOUT THE AUTHOR

KISHORE BHATIA is an engineering director and the head of engineering at BCG Digital Ventures’ New York office. He focuses primarily on blockchain technology–enabled start-ups, among other digital-native ventures. Contact him at bhatia.kishore@gmail.com.