An Asynchronous Panel Discussion

What Are Cloud-Native Applications?

Dennis Gannon

Roger Barga

Neel Sundaresan

During the spring and summer of 2017, the guest editors of this special issue conducted an asynchronous panel discussion among several experts concerning the nature of cloud-native applications. The experts have each been deeply involved in building applications and infrastructure, and they are veterans of the transition to cloud-native design:

- Sebastien Goasguen—author and founder of Skippbox and innovator at Bitnami
- Niklas Gustaffson—cloud architect and program manager at Microsoft
- Balan Subramanian—Microsoft principal program manager in cloud and enterprise focused on cloud tools
- Cornelia Davis—senior director of technology, Pivotal Software, Inc.
- Dan Kohn—executive director of the Cloud Native Computing Foundation

We organized this around responses to six questions that we feel are the most important for defining cloud-native applications. The questions and responses are below.

1. Taking advantage of cloud computing requires a new way of thinking about how applications are designed, systems are architected, and capacity is planned. What are the top three considerations in designing a cloud-native application?

Sebastien: Instead of considerations, I would like to talk about misconceptions. When talking about cloud-native apps and technologies being used to enable these apps (e.g., containers, docker, Kubernetes), some folks tend to think that the app will scale automatically without setup, that databases will get replicated magically, that their data is available everywhere. This has dramatic consequences, if your app is a single process with local data; unreplicated, even running it on AWS is not going to make it resilient. The fallacies of distributed computing are still hard at work, even in cloud native. The network is unreliable, latency is not zero, bandwidth is not infinite, etc.

Niklas: Cloud-native applications require a new way of thinking about application design, architecture, and capacity planning. The top consideration that

2325-6095/17/$33.00 © 2017 IEEE

I see is how to properly break down the functionality into right-sized microservices. To be agile and scalable, services must be sufficiently small, but to avoid complexity, they need to be sufficiently big.

Balan: A cloud-native application is typically loosely coupled, which enables selective scaling, independent updates, and tolerance to partial failure. Since services can be updated independently, architects need to take that into account and figure out how services calling other services will respond to [application programming interface] API changes. They will need to adapt to a service discovery-based pattern, as instances of each service can come and go. This pattern also introduces new challenges in understanding capacity allocation, because capacity now has to be planned for each portion of the application independently. Each service needs to be managed independently against specific [service-level agreements] SLAs.

Cornelia: The top two considerations in designing cloud-native applications are (1) that they are far more distributed than anything we've seen before and (2) that they are operating in an environment that is constantly changing. In fact, I like to say that cloud-native software is "change-tolerant software." Whether the changes are planned (an upgrade to the operating system when a critical vulnerability is found) or unplanned (the failure of an entire hardware rack), your software is expected to be functioning through all of those changes, 24-7. I like that Sebastien called out the fallacies of distributed computing—the number of distributed components in cloud-native software is orders of magnitude greater than what we've seen in software from previous eras, and with that comes a commensurate level of disruption that the fallacies can cause. Thinking of your software as a distributed system is essential.

I'd also like to pile onto the topic of capacity planning—in the cloud, I consider this a bit of an oxymoron. In a world where things are always changing, we shift to a model where we are *managing* capacity instead of trying to *predict* or *plan* it. We need to design our software so that it can grow and shrink with changing request volumes.

Dan: Cloud native—orchestrating containers as part of a microservices architecture—is a departure from traditional application design. Cloud-native technologies like Kubernetes enable higher-velocity software development at a lower cost than traditional infrastructure. Focus on the three properties that make up cloud native: containerized, dynamically orchestrated, and microservices oriented.

2. What kind of workloads are you seeing being built cloud native? What should an organization consider moving first to the cloud?

Sebastien: That sounds like two different questions. You can move a traditional workload to the cloud by using traditional configuration management tools and virtual machines. Cloud-native applications are microservices based [and] tend to have many stateless layers that are set up to scale easily, but more importantly, cloud-native applications are developed much faster and are deployed continuously. So, you can move a monolith to the cloud, but what we are seeing is new services being built with a microservice mindset and being deployed to strangle the monolith. The use of containers, as well as new software development practices, is enabling companies to innovate and design their new applications with a cloud-native mindset from the ground up.

Balan: Any workload can benefit from [the] loose coupling inherent in cloud-native apps, but its easiest to adapt this pattern for workloads where data can be partitioned and localized to each service participating in the cloud-native app and where services lend themselves to be largely stateless. Mobile backends, e-commerce applications, apps with data-crunching workers, etc., are all examples of apps that can be built cloud native.

Dan: Two kinds: greenfield and brownfield. Greenfield means that for new applications, it is easy to start containerized, split your app into parts, and orchestrate them. Brownfield is all of the existing monolithic applications that most business relies on today. There, you can containerize your monolith and then start splitting off chunks of functionality over time, until you eventually whittle it down to a reasonable size.

3. What changes in engineering practices and what new roles appear in an engineering team when building and managing cloud-native applications?

Sebastien: The biggest change over the last 5, maybe even 10, years has been the concept of continuous deployment. This is a natural by-product of cloud technologies that are API driven and have enabled lots of software development life cycle tasks to be automated, from the packaging to the testing and delivery. The entire cycle has been sped up dramatically, [and] this has given rise to the [development and operations] DevOps movement, which is a community of practice around these ideas: complete automation of the pipeline from development to production. This evolution will continue as more apps start to use cloud service directly and focus on the real business logic and value. Function as a service is most likely set to be the next evolution that will change engineering practices even further.

Balan: The key change is that each engineering team has to do its own DevOps on the services it builds and operates. Centralized [continuous integration and continuous delivery] CI/CD becomes a bottleneck to adopting cloud-native apps, so teams will take this on themselves. They will need to add roles to manage service integrations; monitoring and maintaining SLAs, managing API changes, etc., will need more attention.

Cornelia: In the cloud, tickets have been replaced by self-service, with developers and operators now obtaining the resources they need by invoking an API. This has enabled the type of automation that drives the continuous deployment that Sebastien has described. But then there needs to be a system that provides implementations for those APIs, and such a platform, when well designed, allows us to separate the concerns of the *application team* from [those of] the *platform team*. Members of the latter are proficient with the abstractions that have dominated [information technology] IT for decades—compute, storage, and network—and they provide a self-service platform that projects higher-level abstractions—functions, apps, and services. Members of app teams leveraging that platform are then relieved of the burdens of managing infrastructure and can focus on providing business value through the software they write.

Dan: CI/CD is an essential change in how organizations operate. Going from a deploy once a quarter or once a month to dozens of deploys per day means that your organization becomes far more responsive to your customers' needs.

4. Many cloud-native apps are built around a microservice architecture. Are there examples of cloud-native designs that are not microservice based? What do they look like?

Sebastien: Cloud native is really a loaded term, like cloud was at one point, like serverless is becoming. I like to think in terms of evolution. Technology evolves, which influences our software practice. In some sense, microservice is really [service-oriented architecture] SOA; you are trying to break up applications so that they are easier to develop, such that each component can run on the network, and so on. You could say that microservices are to apps what threads are to processes, and that's why systems like Kubernetes are sometimes referred to as the Linux of the cloud. So what would be a cloud-native design that is not based on microservices? I suppose it would be a monolith that is aware of the cloud API it's running on and can interact with it: to get metadata, to trigger other actions, to use other services. I have not run into any of those really.

Balan: Cloud-native apps are by definition microservices based. However, before microservices became mainstream, developers had been building loosely coupled apps that use managed services for data and communicate with other app components through events placed on a queue.

Cornelia: While I do believe microservice architectures are your best bet if you aim to build cloud-native applications, I also don't think it needs to be as strictly defined as that. First, we have to acknowledge that the terms we're dealing with here are relatively vague. Key criteria of what makes a microservice architecture could include the size (as in number of lines of code) of the services, whether each service has its own source code repository, whether a solution is composed of many independent services or relatively few, or even whether services are deployed as singletons or as multiples. And the definition of cloud-native apps is equally fuzzy. I personally define the latter as applications that run well in the highly distributed, constantly changing environments that make up the cloud (another term that can be interpreted a number of different ways—i.e., public cloud vs. private). Some might argue that Etsy is an application that runs very well in [its] private cloud, yet famously, the Esty architecture is not microservices based. It is based on a traditional software stack consisting of Linux, Apache, MySQL, and PHP with load balancing and caching sprinkled throughout, and I've even heard Etsy engineers refer to their application as monolithic. But it does have some modern elements too, like Apache Kafka, that they use for some event-driven use cases. Whether we call them cloud native or microservices based, what is most important is that developers understand the patterns that afford software the resilience and agility that it needs today.

Dan: Monolithic apps that are evolving to be cloud native may not start out microservices based, but they can evolve there by splitting off pieces of functionality and writing new functionality in separate apps.

5. Resilience and adaptability are said to be by-products of cloud-native design. Why is this the case? How can I measure it?

Sebastien: I will just give an example. When I first encountered Kubernetes, I was shocked because it was a system that was tough to kill. You could kill the API server, shut down the key-value store, and your app would still be running. You can shut down nodes in your cluster, and your app would still be running. A few simple concepts allowed the application to be fault tolerant: replication controllers and the restart of the containers. So, if you write your applications using the Kubernetes API objects, you will benefit from all of this and more. I am not sure what you mean by adaptability, but I will take another example. Kubernetes has cluster federation built in. This means that you can define a service and Kubernetes will automatically replicate it across clusters. Assuming that your networking (aka load balancers) is in place to route traffic to your app, if one of your clusters goes down, your app will not end automatically; it will scale to adapt to the failed cluster.

Niklas: The way I see it, the observed resilience and adaptability of cloud-native applications is first and foremost a consequence of adopting a microservices-based architecture. The ability to easily take advantage of multiregion compute and storage capabilities contributes significantly to the resilience of cloud-native apps.

Balan: Since the services in the app are loosely coupled, a single service going down does not bring down the complete app but makes only some features of the app unavailable. The same goes for application updates, as new features can be rolled out without bringing down other features. Teams taking advantage of the cloud-native pattern should expect an increase in the overall availability of their applications and reduced planned and unplanned downtime.

Dan: Essentially, every big company that has needed to reach webscale has independently developed a microservices architecture, because no single application or server can provide the necessary resilience. Instead, you need to assume that individual parts of your application will fail and design an architecture that nevertheless provides resiliency.

6. In 5–10 years, where do you expect the world of cloud and cloud-native development to be?

Sebastien: There is still a ton of workloads that are running [on premises] in the enterprise. But when you start from scratch, it is becoming increasingly hard not to use cloud services. Whether you need a traditional relational database or you want to take advantage of a cutting-edge service for machine learning, the cloud has an ever-increasing appeal when writing new applications. This has a direct consequence on the software practice. I think we will see developers and even operators not wanting to run their own services and instead focus on the core value of their applications. And that's where serverless comes in. This is all coming full circle; computing is finally a utility (certainly infrastructure is), basic components are a utility (e.g., database, archives, load balancers), so what does that mean? It means that we will get back to writing application code [for] applications that are much closer to the user and the core business.

Balan: We believe that services would be more intelligent. For example, they would be able to scale themselves when they notice that they are becoming a bottleneck to a calling service and they will be able to adapt their client code as APIs of services they call evolve.

Niklas: The cloud space is moving too fast for anyone to make meaningful predictions 5–10 years out. Don't believe anything anyone says about cloud in 2022 or 2027!

Dan: I expect the cloud-native software stack, built around Kubernetes, to provide portability between public, private, and hybrid cloud architectures. There will be tons of competition below the stack (who provides your servers) and above (who provides data stores and value-added services), but I think there will be remarkable convergence in the middle as Kubernetes becomes the Linux of the cloud.

SEBASTIEN GOASGUEN *is a 15-year open-source veteran. Former associate professor of computer science at Clemson University, he worked on Apache CloudStack for several years before devoting himself to containers and cloud-native applications. He founded Skippbox, a Kubernetes start-up acquired by Bitnami. He is now the senior director of cloud technologies for Bitnami, overseeing all Kubernetes efforts. He recently created kubeless, a serverless solution built on top of Kubernetes. Goasguen has written more than 70 international articles. He speaks frequently at open-source events and is the author of the O'Reilly* Docker Cookbook *and* Kubernetes Cookbook. *Contact him at runseb@gmail.com.*

NIKLAS GUSTAFFSON *is a principal program manager in Microsoft's Developer Division and has been with Microsoft for 15 years. He's most recently been involved with technology and tooling for Azure software development kits. Before that, Niklas was involved with several Developer Division efforts, including Visual Basic, parallel computing, and distributed computing libraries. Before working for Microsoft, Niklas was involved with a series of start-ups. He started his career at Digital Equipment Corp. He holds a MS in computer science from the Royal Institute of Technology in Stockholm, Sweden. Contact him at niklas.gustafsson@microsoft.com.*

CORNELIA DAVIS *is senior director of technology at Pivotal, where she works on the technology strategy both for Pivotal and for Pivotal customers. Through engagement across Pivotal's broad customer base, Cornelia develops core cloud platform strategies that drive significant* *change in enterprise organizations and influence the Pivotal Cloud Foundry evolution. She is working on ways to bring the various cloud-computing models of infrastructure as a service, application as a service, container as a service, and function as a service together into a comprehensive offering that allows IT organizations to function at the highest levels. She is the author of the book* Cloud Native: Designing Change-Tolerant Software *by Manning Publications (https://www.manning.com/books/cloud-native). An industry veteran with almost three decades of experience in image processing, scientific visualization, distributed systems, web application architectures, and cloud-native platforms, Cornelia holds a BS and an MS in computer science from California State University, Northridge, and further studied theory of computing and programming languages at Indiana University. Contact her at cornelia@corneliadavis.com.*

BALAN SUBRAMANIAN *leads the product team building developer experiences for Azure through a combination of tools and cloud services. He is passionate about helping polyglot developers build cloud-native apps with productive workflows that help them deliver value to customers faster. Previously, Balan led product management at Heroku and AWS, where he built platform services that enabled developers to collaborate better, adopt new application patterns, and embrace modern workflows. Contact him at balans@microsoft.com.*

DAN KOHN *is executive director of the Cloud Native Computing Foundation, a Linux Foundation project and organization advancing the development of cloud-native technologies. He also helped create and launch the Linux Foundation's Core Infrastructure Initiative. As the no. 2 person at the Linux Foundation, Dan previously managed the not-for-profit trade group. Collaborating closely with the high-powered board representing the whole technology industry, Dan rebuilt the Linux Foundation into one of the top industry consortia. He has continued to assist the foundation, including in the development of the Node.js Foundation and the blockchain-focused Hyperledger Foundation. Contact him at dan@dankohn.com.*