Containers vs. VMs vs. Serverless at the Edge

February 26, 2019

Edge

The rise of edge computing has brought about a shift in system architecture requirements and considerations. As applications demand lower latency and reduced bandwidth, deployment method decisions are increasingly critical. This article examines the differences between using virtual machines (VMs) vs. containers vs. serverless functions in the context of edge computing, and looks at how to decide which method is best for your workload.

Definitions at the Edge

Let’s first take a look at the definitions for each of these separately, and see how they fit into edge computing.

Virtualization

Cloud computing has its roots in virtualization. A virtual machine is an emulation of a computer system. There are system VMs (or full virtualization VMs), which offer a substitute for a physical machine and process VMs, which execute computer programs in a platform independent environment.

All kinds of virtualization enable the running of your application on fewer physical servers. Through virtualization, each app and operating system live in a separate software container called a virtual machine or VM. VMs are entirely isolated; however, computing resources, storage and networking are pooled together and delivered dynamically to each VM by software called a hypervisor (the software or firmware layer). This means that every app gets what it needs for peak performance and the strong isolation of the system is important for enabling robust security.

With servers running at full capacity, fewer are needed. The idea is that performance improves, less maintenance is required, and savings can be made on hardware and overhead. VMs have formed the foundation of Infrastructure as a Service (IaaS), the most popular delivery model of the cloud. Amazon EC2, Azure VMs, and Google Compute Engine are examples of IaaS.

Containerization

There are inherent advantages to containerization over VMs. Compared to virtualization in which each VM runs a unique operating system atop a hypervisor, containers run on top of physical infrastructure. Hypervisors and OS are reduced to wafer thin layers that act as an interface between the hardware and containers.

Containers can be a strong candidate for running edge workloads because they are very lightweight and agile as a result of sharing the same kernel. The shared OS also reduces the number of management tasks, such as patching and upgrades for the OS admin.

Containers can be run on virtual machines or bare metal. CenturyLink recently ran a test to determine the network latency of Kubernetes clusters running on both VMs and bare metal, and found that containers running on bare metal servers achieved three times lower latency compared to when running Kubernetes on VMs. CPU consumption was also higher when a cluster was run on VMs.

Serverless

Following VMs and containerization, the next wave of compute services is serverless, also known as Functions as a Service (FaaS). The goal behind serverless is to make the developer experience more straightforward by reducing the operational overhead in running and managing code. FaaS platforms such as AWS Lambda, Azure Functions and Google Cloud Functions are used as the foundation for contemporary applications designed as microservices.

As the unit of deployment in serverless is a function, it is a significantly more efficient solution than a comparatively heavy VM or container.

In terms of edge compute, serverless platforms can play a particularly important role in simplifying the DevOps cycle, particularly in relation to operations involving resource intensive compute, such as machine learning (ML).

Use Cases Along the Edge Continuum

VMs: VNFs and CaaS

VMs are often used as the basis for network services. Many vendors have chosen not to containerize because of the level of isolation offered by VMs and the robust security this enables, both in terms of web application firewalls (WAFs) and other security services.

Various tasks can be run in VMs via virtual network functions (VNFs) (virtualized tasks previously carried out by proprietary, dedicated hardware). VNFs operate on software that runs on commodity hardware. The kinds of tasks they perform include WAFs, caching, DNS or network address translation. VNFs want to look at every single packet, which containers currently can't do, although network service mesh may be a solution for containerization to be able to achieve this.

AWS Greengrass, a service that extends Amazon Web Services functionality to edge compute, is an example of a Container as a Service (CaaS). Built on top of an existing infrastructure layer based on VMs, Greengrass delivers portability and agility, two necessities of edge computing. If Greengrass loses connectivity from the cloud, it can still communicate with services locally through Greengrass core and provide messaging and security services for local execution of AWS Lambda. IoT SDK enabled devices can also communicate with Greengrass core via the edge network.

Containers: Retail in a Box

The restaurant Chick-fil-A has begun to use bare metal clustering for Kubernetes at the edge in its restaurants. This equates to approximately 6,000 devices at the edge running Kubernetes in 2,000 restaurants. This gives the restaurant's edge deployment a unique kind of scale. Instead of having only a few large K8s clusters with tens-to-hundreds-of-thousands of containers, at full scale, it has over 2,000 clusters with tens of containers per cluster.

Chick-fil-A's edge workloads include platform services (such as log collection, monitoring, pub/sub messaging), applications that interact with "things" inside each restaurant, and ML models that are able to synthesize cloud-developed forecasts with real-time events from pub/sub messaging in order to issue decisions at the edge and drive automation processes.

Why has Chick-fil-A sought out this kind of deployment? In a follow-up post, the IoT/edge team explained a variety of objectives, including "low-latency, Internet-independent applications", which "can reliably run our business". The others mentioned comprise of "high availability for these applications", a "platform that enables rapid innovation and that allows delivery of business functionality to production as quickly as possible", and the opportunity to achieve a "horizontal scale" in relation to its infrastructure and app development teams.

Use cases for containerization are more extensive than serverless mainly because you can refactor existing monolithic applications to container-based setups. Further uses of containerization include web APIs, machine learning computations and long-running processes.

Serverless: Machine Learning at the Device Edge (IoT)

Machine Learning (ML) models are being used throughout the process of developing software. Increasingly, machine learning models are being deployed at the edge (closer to where data is generated). By operating and gaining insights as close to the source as possible, machine learning models deployed at the edge can reduce transmission costs, reduce application latency, and isolate privacy concerns.

In autonomous vehicles, for instance, a predictive maintenance model could be kept near the automobile as opposed to running in the cloud. By invoking the ML model closest to the application, latency is dramatically reduced versus sending data on a round trip to and from the cloud.

Additional use cases for serverless at the edge include performing back-end tasks for mobile apps or websites, thereby freeing up front-end time and resources; cleaning up, parsing and filtering data streams and uploads from real-time devices, thus moving resource-intensive processes out of the main application; and high volume background processes such as migrating data to long-term storage and forwarding metrics to an analytics service.

The DevOps POV

From a DevOps standpoint, there are pros and cons to all three methods. The two most important factors to consider are development speed and time-to-market. Other factors to weigh are the complexity of your deployment scenario, the time needed to deploy an application plus vendor lock-in and associated costs.

In summary, containerization can be a powerful tool when you need flexibility and total control of your system, or when legacy services need migrating. Tools for monitoring and proper alerts such as Prometheus, have also matured with containerization.

The main drawback of containers from a DevOps point of view is that having more control and greater flexibility also means greater complexity and more moving parts. It also introduces greater costs as you will be paying for resources all the time, whether there is traffic or not.

Furthermore, with containerization, scaling is not enabled by default. This can mean a more complicated setup process; however, it also means the developer benefits from full control of his/her resources as you are entirely in charge of the scaling, which (depending on the provider) theoretically means infinite scalability.

Serverless, on the other hand, can be particularly useful if you have sudden spikes in traffic that need to be immediately detected and handled. You pay only for the resources you use, so if there is no traffic, the application will be shut down. Autoscaling is enabled by default.

Furthermore with serverless, the infrastructure is abstracted away from the developers creating the function, making it an ideal solution for using serverless databases such as DynamoDB and Aurora Serverless, hosting static websites on S3, and running code without the hassle of managing servers. Serverless eliminates the work involved in deploying your applications (you simply have to deploy code to your provider) and the administration involved in installing OS updates or making security patches. This means fast, flexible time-to-market, an extremely valuable asset for any startup.

Serverless is a powerful tool for microservices frameworks, such as web APIs, task runners or the processing of data streams and/or images. For background tasks that only run occasionally, such as Cron jobs, serverless is a good choice. In fact, FaaS is only viable for short-running processes (the maximum time an AWS Lambda function can run is five minutes). For heavier computing tasks, a container or VM based setup would be preferable.

Other downsides to serverless include the fact that there are predefined limits for memory and processing power, calling for extremely efficient code to avoid overloading functions if they get too large, which can lead to latency challenges. Serverless is also trailing behind in terms of the maturity of its dev tools.

Virtualization is the oldest technology of the three and although VMs have performed a great job over the last decade or more, containerization and serverless technologies are largely superceding them in terms of running edge workloads. That said, there are various developments in infrastructure technologies that are seeking to combine VMs and containers, such as New York-based startup Hyper. HyperContainers are an attempt to provide the best of both worlds by offering the speed and agility of containers with the security for, and isolation of the VM.

Security Concerns

Previously, customers have preferred VMs to containers for security reasons due to their more complete isolation. However, VMs have their own set of security concerns that need addressing. One simple way of improving security is to limit running services to only what is absolutely necessary. Another is to apply patches as quickly as possible following their release. On a full virtual machine, the need to apply any given patch is usually far higher than in containers and serverless as there are substantially more packages required and installed.

Container-specific security concerns can be focused around two specific areas: the trustworthiness of the container source on which the container is built, and the level of access the container is allowed to the host operating system. The container should never be run with root or administrator privileges when running on any host, whether Windows or Linux. Moreover, containerization carries all the same security concerns as serverless.

Security issues related to serverless include ensuring that anyone with access follows secure coding best practices. This is especially important in FaaS as any vulnerability will lead to leaked data that can easily spill beyond the scope of the serverless app itself. The other primary security issue in serverless is related to any third-party library included inside the app. They should always be used in concert with a scanning tool that self-updates and routinely scans your built artifacts or a highly organized manual process in order to stay on top of any vulnerability announcements related to any third-party library used.

In Conclusion

Ultimately, the decision is not whether to use one over the other. Depending on your workload requirements, each has their own benefits and can be mutually supportive within computing architectures. Focus on what your workload demands first, then decide which solution is best to solve that problem.