Serverless vs Containers: Choose One or Use Both?

Understand the key differences between serverless and containers, and whether you should use one or both in your next cloud native project

What are Containers?

Containers allow applications to be packaged, separated from the underlying infrastructure, and deployed consistently in any environment. Containers share access to the host machine's operating system kernel, and do not require running an entire operating system like in a full virtual machine (VM). Containers can be deployed on bare metal servers, in cloud VMs or specialized container instances, or via managed services.

Container engines run containers based on images, which specify exactly which applications, configuration and data the container should contain. Containerized applications can be constructed using multiple container images. For example, an application could consist of a web server, application server, and database, each running in a separate container.

Containers are by nature stateless, and cannot store state information beyond their lifetime (once the container is shut down, the state is lost). But they can be used for stateful applications, for example by storing persistent data to an external storage volume.

What is Serverless?

Serverless computing is a computing paradigm in which computing resources are managed behind the scenes, and developers focus on writing code to handle specific events. This code is packaged as a serverless function, and invoked by the serverless runtime as many times as necessary to serve incoming requests.

In a traditional infrastructure as a service (IaaS) model, the cloud provider offers a virtual machine and bills for the time it is used, regardless of workloads actually running on the virtual machine. The customer is responsible for running the virtual machine, deploying workloads on it, and scaling it up or down as necessary.

By contrast, in a serverless architecture, the customer is only responsible for providing the serverless function, and is not aware of the underlying compute resources. The serverless runtime provisions server resources automatically, and customers are billed according to the number of times and the duration their function actually ran.

In this article, you will learn:

Serverless Computing and Containers: What's the Difference?

- Save the overhead and complexity of virtual machines
- Abstract applications from the underlying host environment
- Automate and dynamically scale workloads

However, in other respects, serverless and containers are quite different. The following table lists the differences.

	Containers	Serverless
Supported host environments	Containers can run on modern Linux servers and certain Windows versions.	Serverless runs on specific hosting platforms, most of which are based in public clouds, such as AWS Lambda and Azure Functions.
Running locally	Containers can easily be run in a local data center or on a developer's workstation.	Serverless is more difficult to run outside a public cloud environment. Local serverless frameworks do exist, but are still complex and not widely adopted.
Cost	The majority of container engines and orchestrators are open source, and you can run them in a local environment for free (taking into account the time needed to deploy and maintain them).	Serverless environments are hosted in the public cloud and are billed per use.
Supported languages	Applications can be containerized as long as the underlying host server supports the language they are written in.	To run an application in a serverless model, the serverless runtime must explicitly support that language (different platforms support different languages).
Statefulness	Containers are stateless by nature, but it is possible to set up persistent storage to support stateful applications.	Most serverless runtimes are designed to support stateless workloads. Some serverless providers have limited support for stateful services. Serverless runtimes also make it possible to connect to cloud storage services to persist data.
Availability	Containers can run for prolonged periods of time.	Serverless functions typically run for a short period of time (minutes or seconds) and are shut down as soon as they finish processing the current event or data.

Containerization vs Serverless: Choose One or Use Both?

Let's consider the relative advantages of containerized vs. serverless applications. Of course, the best choice will depend on your use case, team skill set, and specific requirements. In many cases, containers and serverles can work together, leveraging the relative benefits of each technology. Read more in the following section.

Containerization Benefits

To get the maximum benefit from containerization, you need to operate in a microservices model. Split your application into individual, independent services, each of which can be deployed as a container, and use an orchestrator like Kubernetes to manage these containers as a cluster.

Kubernetes provides powerful capabilities such as auto scaling, fault tolerance, storage and networking management.

However this flexibility comes at a cost of higher complexity. Container orchestrators are complex to deploy and use, and require specialized expertise. Managing a large containerized application can be a full time job—or several full time jobs. In addition, costs can quickly get out of control, especially when running in the public cloud.

Serverless Benefits

Serverless is best if you need to perform relatively simple processing on streams of events. It is easy to set up, even at large scale, and you pay only for the exact time your serverless functions are in action. It is even more convenient when the event source is a service running on the same cloud provider.

Unlike containerization, there is no infrastructure to manage. You only have to worry about your code and the business value it provides.

However, there are currently some limitations with respect to vendor support and ecosystem dependencies. Supported programming languages and runtime environments are limited by what the serverless provider supports. There are some workarounds for these limitations.

Due to the high abstraction of the infrastructure, it is difficult to monitor and visualize how applications are running. Serverless applications, like any other applications, have performance and stability issues, and it can be much more difficult to debug and resolve these issues, increasing risk for production deployments.

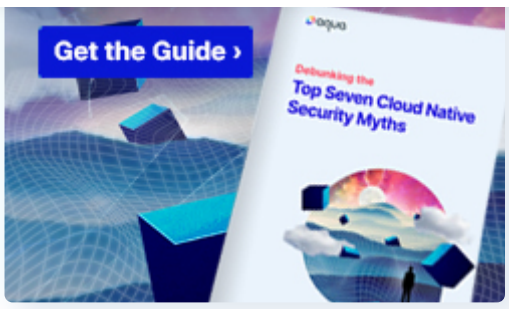
Can Serverless and Containers Work Together?

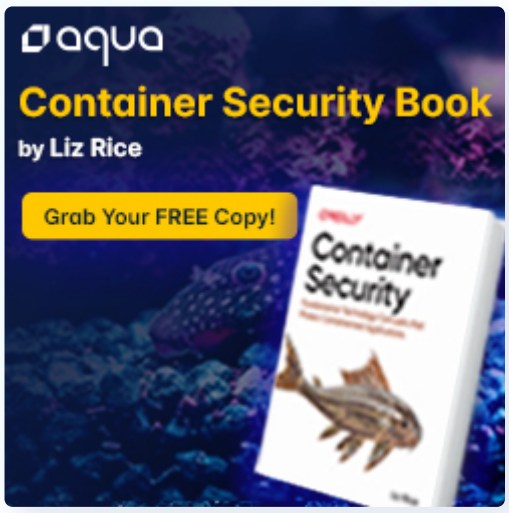
The relative advantage of serverless and containers can compensate for each other's weaknesses. Combining these two technologies can be very useful.

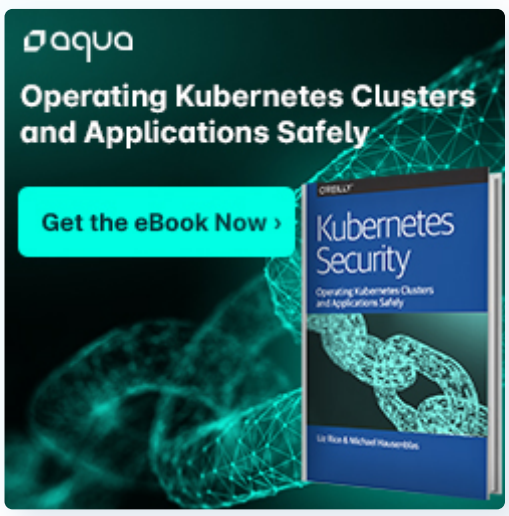
If your application uses a monolithic architecture, and is too large to run on a serverless runtime, this doesn't mean you can't leverage serverless. Many applications have small back end tasks, typically implemented using chron jobs, which are packaged with the application when it is deployed. These tasks are a classic fit for serverless functions.

Similarly, if you have a complex containerized system, and are running certain ancillary tasks triggered by events, don't run them in a container. Separate those tasks to a serverless function to offload complexity from your containerized setup, and enjoy the simplicity and cost effectiveness of serverless.

In the opposite direction—you can easily expand a serverless application using containers. Serverless functions typically persist data to cloud storage services, and you can mount these services as Kubernetes persistent volumes. This allows you to integrate and share stateful data between serverless and container architectures.







Aqua Blog
All about cloud native and security