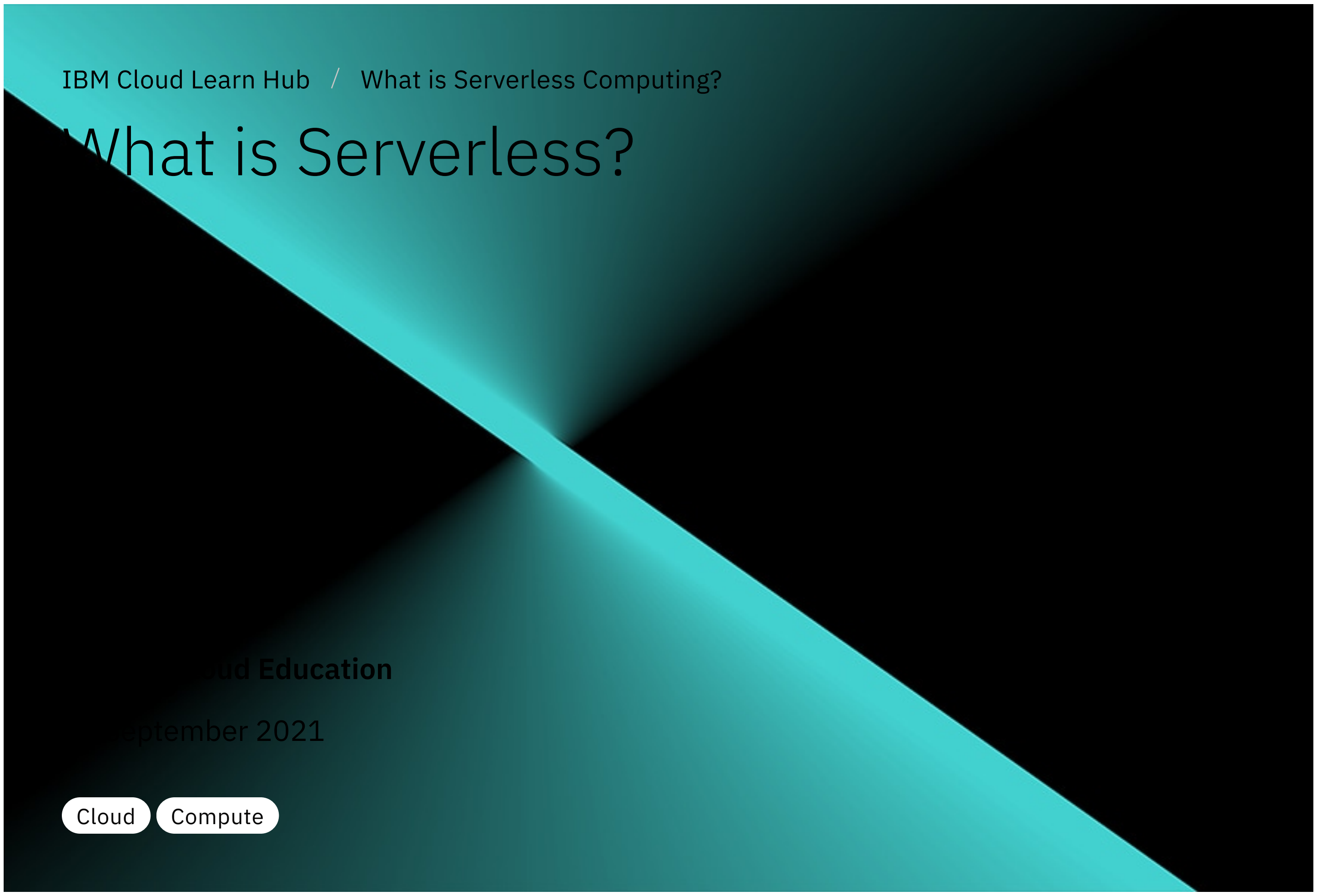


IBM Cloud Learn Hub / What is Serverless Computing?

What is Serverless?

[illegible]ud Education

[illegible]eptember 2021

Cloud Compute

What is serverless?

What is Serverless?

Serverless is a cloud application development and execution model that lets developers build and run code without managing servers, and without paying for idle cloud infrastructure.

What is serverless?

Serverless is a cloud computing application development and execution model that enables developers to build and run application code without provisioning or managing servers or backend infrastructure.

Serverless lets developers put all their focus into writing the best front-end application code and business logic they can. All developers need to do is write their application code and deploy it to containers managed by a cloud service provider. The cloud provider handles the rest, provisioning the cloud infrastructure required to run the code and scaling the infrastructure up and down on demand as needed. The cloud provider is also responsible for all routine infrastructure management and maintenance such as operating system updates and patches, security management, capacity planning, system monitoring and more.

Also important: With serverless, developers never pay for idle capacity. The cloud provider spins up and provisions the required computing resources on demand when the code executes, and spins them back down again—called ‘scaling to zero’—when execution stops. The billing starts when execution starts, and ends when execution stops; typically, pricing is based on execution time and resources required.

Serverless does not mean 'no servers'

The name notwithstanding, there are most definitely servers in serverless computing. 'Serverless' describes the developer’s experience with those servers—they are are invisible to the developer, who doesn't see them, manage them, or interact with them in any way.

Today every leading cloud service provider offers a serverless platform including Amazon Web Services (AWS Lambda), Microsoft Azure (Azure Functions), Google Cloud (Google Cloud Functions) and IBM Cloud (IBM Cloud Code Engine). Together serverless computing, microservices and containers form a triumvirate of technologies typically considered to be at the core of cloud-native application development.

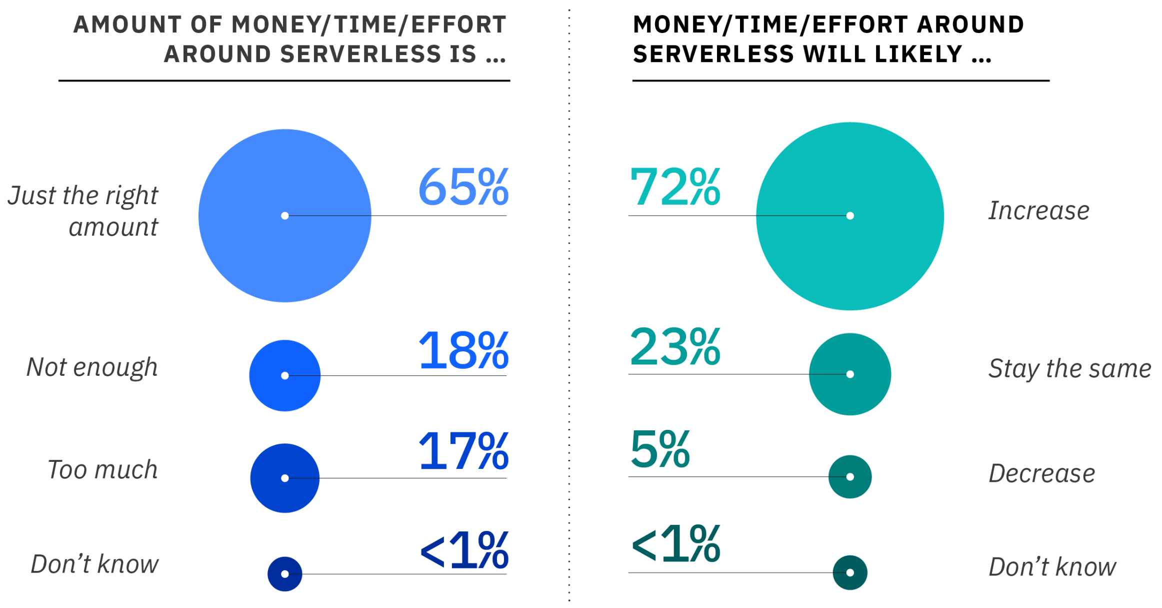


***Figure 1: Organizations will continue to invest in serverless.** 95 percent of users surveyed plan to maintain or increase their serverless investments during the next year. (Source: 'Serverless in the enterprise, 2021' (PDF, 1.8 MB))*

Serverless is more than just FaaS

Function-as-a-service, or FaaS, is a cloud computing service that enables developers to run code or containers in response to specific events or requests, without specifying or managing the infrastructure required to run to code.

FaaS is the compute model central to serverless, and the two terms are often used interchangeably. But serverless is much more than FaaS. Serverless is an entire stack of services that can respond to specific events or requests, and scale to zero when no longer in use—and for which provisioning, management and billing are handled by the cloud provider and invisible to developers. In addition to FaaS, these services include:

- **Serverless databases and storage**: Databases (SQL and NoSQL) and storage (particularly object storage) are the foundation of the data layer. A serverless approach to these technologies involves transitioning away from provisioning "instances" with defined capacity, connection and query limits, and moving toward models that scale linearly with demand in both infrastructure and pricing.

- **Event streaming and messaging**: Serverless architectures are well-suited for event-driven and stream-processing workloads most notably the open source Apache Kafka event streaming platform.

- **API gateways**: API gateways act as proxies to web actions and provide HTTP method routing, client ID and secrets, rate limits, CORS, viewing API usage, viewing response logs, and API sharing policies.

IBM's Ashher Syed provides a detailed explanation of serverless and and the serverless stack (6:37):

Serverless vs. PaaS, containers, and VMs

Because serverless, platform as a service (PaaS), containers, and virtual machines (VMs) all play a critical role in the cloud application development and compute ecosystem, it's useful to compare how serverless compares to the others across some key attributes.

- **Provisioning time**: Measured in milliseconds for serverless, vs. minutes to hours for the other models.

- **Administrative burden**: None for serverless, compared to a continuum from light to medium to heavy for PaaS, containers and VMs respectively.

- **Maintenance**: Serverless architectures are managed 100% by the provider. The same is true for PaaS, but containers and VMs require significant maintenance including updating/managing operating systems, container images, connections, etc.

- **Scaling**: Auto-scaling—including auto-scaling to zero—is instant and inherent for serverless. The other models offer automatic but slow scaling that requires careful tuning of auto-scaling rules, and no scaling to zero.

- **Capacity planning**: None needed for serverless. The other models require a mix of some automatic scalability and some capacity planning.

- **Statelessness**: Inherent for serverless, which means scalability is never a problem; state is maintained in an external service or resource. PaaS, containers and VMs can leverage HTTP, keep an open socket or connection for long periods of time, and store state in memory between calls.

- **High availability (HA) and disaster recovery (DR)**: Both are inherent in serverless with no extra effort and at no additional cost. The other models require additional cost and management effort. In the case of both VMs and containers, infrastructure can be restarted automatically.

- **Resource utilization**: Serverless is 100% efficient because there are no such things thing as idle capacity—it is invoked only upon request. All other models feature at least some degree of idle capacity.

- **Billing granularity and savings**: Serverless is metered in units of 100 milliseconds. PaaS, containers and VMs are typically metered by the hour or the minute.

Serverless, Kubernetes and Knative

Kubernetes is an open-source container orchestration platform that automates the deployment, management and scaling of containers. Kubernetes' automation dramatically simplifies the development of container-based applications.

Serverless applications are often deployed in containers. But on its own, Kubernetes can't run serverless apps without specialized software that integrates Kubernetes with a specific cloud provider's serverless platform.

Knative provides a serverless framework for Kubernetes. It’s an open-source extension to Kubernetes that enables any container to run as a serverless workload on any cloud platform that runs Kubernetes, whether the container is built around a serverless function or some other application code (e.g., microservices). Knative works by abstracting away the code and handling the network routing, event triggers and autoscaling for serverless execution.

Knative is transparent to developers—they just build a container as usual using Kubernetes, and Knative does the rest, running the container as a serverless workload.

Learn more about Knative

Pros and cons of serverless

Pros

Given all of the preceding, it should be no surprise that serverless computing offers a number of technical and business benefits to individual developers and enterprise development teams.

Improved developer productivity: As noted above, serverless enables development teams to focus on writing code, not managing infrastructure. It gives developers much more time to innovate and optimize their front-end application functionality and business logic.

Pay for execution only: The meter starts when the request is made, and ends when execution finishes. Compare this to the infrastructure as a service (IaaS) compute model, where customers pay for the physical servers, virtual machines (VMs) and other resources required to run applications, from the time they provision those resources until the time they explicitly decommission them.

Develop in any language: Serverless is a polyglot environment, enabling developers to code in any language or framework—Java, Python, JavaScript, node.js—with which they're comfortable.

Streamlined development/DevOps cycles. Serverless simplifies deployment and, in a larger sense, simplifies DevOps because developers don't spend time defining infrastructure required to integrate, test, deliver and deploy code builds into production.

Cost-effective performance. For certain workloads—embarrassingly parallel processing, stream processing, certain data processing tasks—serverless computing can be both faster and more cost-effective than other forms of compute.

Usage visibility. Serverless platforms provide near-total visibility into system and user times and can aggregate usage information systematically.

Development and IT professionals cite other specific benefits of serverless computing. You can explore them using the interactive tool below:

See what users are saying about serverless architecture.

Survey results show the average percent of users that rate each benefit as the most important for their teams.

Learn more about each benefit

Source: 'Serverless in the enterprise, 2021' (PDF, 1.8 MB)

Cons

With so much to like about serverless, organizations are using it for a wide variety of applications (see Figure 2 below). But there are drawbacks—some of which are related to specific applications, and others which are universal.

Unacceptable latency for certain applications: Because serverless architectures forgo ongoing processes in favor of scaling up and down to zero, they also sometimes need to start up from zero to serve a new request. For many applications the resultant delay would not be detrimental or even noticeable to users. But for others—say, a financial trading application—this cold-start latency might be unacceptable.

Higher costs for stable or predictable workloads: Because serverless scales up and down on demand in response to workload, it offers significant cost savings for spiky workloads. But it does not offer the same savings for workloads characterized by predictable, steady or long-running processes; in these cases, a traditional server environment might be simpler and more cost-effective.

Monitoring and debugging issues: These operational tasks are challenging in any distributed system, but serverless architecture (or microservices architecture, or a combination of the two) only exacerbates the complexity. For example, teams may find it difficult or impossible to monitor or debug serverless functions using existing tools or processes.

Vendor lock-in: As noted, one of the biggest advantages of serverless is that the cloud provider manages all the computing resources. While this frees up considerable time for developers to focus on writing and improving their code, it also means that migrating code to a new cloud provider could prove challenging. Many cloud provider's serverless platforms are designed to provide an ecosystem of managed cloud services and are not portable like virtual machines (VMs) or Docker containers. Application code that triggers several services offered by one cloud provider's serverless platform might have to be partially or completely rewritten to get get the same results in another provider's platform.

Use cases for serverless

Given its unique combination of attributes and benefits, serverless architecture is well-suited for use cases around microservices, mobile backends, and data and event stream processing.

Serverless and microservices

The most common use case of serverless today is supporting microservices architectures. The microservices model is focused on creating small services that do a single job and communicate with one another using APIs. While microservices can also be built and operated using either PaaS or containers, serverless has gained significant momentum given its attributes around small bits of code, inherent and automatic scaling, rapid provisioning, and a pricing model that never charges for idle capacity.

API backends

Any action (or function) in a serverless platform can be turned into a HTTP endpoint ready to be consumed by web clients. When enabled for web, these actions are called web actions. Once you have web actions, you can assemble them into a full-featured API with an API gateway that brings additional security, OAuth support, rate limiting, and custom domain support.

For hands-on experience with API backends, try the tutorial “Serverless web application and API.”

Data processing

Serverless is well-suited to working with structured text, audio, image, and video data, around tasks such as data enrichment, transformation, validation, cleansing; PDF processing; audio normalization; image processing (rotation, sharpening, noise reduction, thumbnail generation); optical character recognition (OCR); and video transcoding. For a detailed image process use case, read “How SiteSpirit got 10x faster, at 10% of the cost.”

Massively parallel compute/“Map” operations

Any kind of embarrassingly parallel task is a good use case for a serverless runtime, with each parallelizable task resulting in one action invocation. Sample tasks include everything from data search and processing (specifically Cloud Object Storage), Map(-Reduce) operations and web scraping to business process automation, hyperparameter tuning, Monte Carlo simulations and genome processing.

For a detailed example, read "How a Monte Carlo simulation ran over 160x faster on a serverless architecture vs. a local machine."

Stream processing workloads

Combining managed Apache Kafka with FaaS and database/storage offers a powerful foundation for real-time buildouts of data pipelines and streaming apps. These architectures are ideally suited for working with all sorts of data stream ingestions (for validation, cleansing, enrichment, transformation), including IoT sensor data, application log data, financial market data and business data streams (from other data sources).

Common applications for serverless

In a recent IBM survey, IT professionals reported using serverless across a wide range of applications, including customer relationship management (CRM), analytics and business intelligence, finance and more:

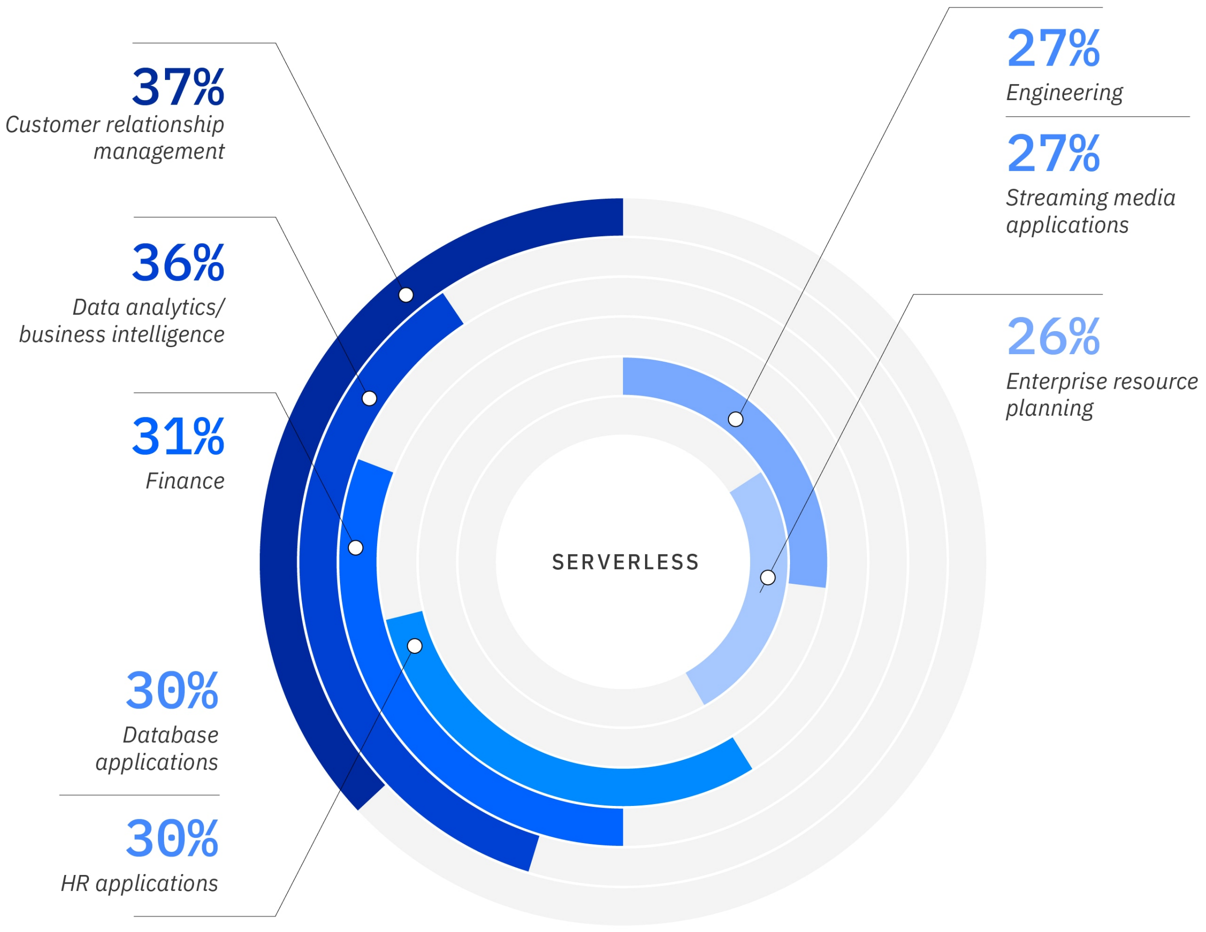


Figure 2: How serverless is being used. *Survey respondents identified over a dozen serverless applications in use. The most commonly cited applications included CRM, data analytics/business intelligence, finance, database, HR, engineering, streaming media and ERP. (Source: Source: 'Serverless in the enterprise, 2021'* (PDF, 1.8 MB)*')*

Tutorials: Get started with serverless computing

You can expand your serverless computing skills with these tutorials:

- Getting started with IBM Cloud Code Engine: Visit our "Hello world" tutorial to see for yourself how easy it is to create and deploy an IBM Cloud Code Engine application.

- Build a container image from source with the Code Engine CLI: The build process uses the buildpacks strategy and stores the image from the build process on Docker Hub. Code Engine supports building from a Dockerfile and Cloud Native Buildpacks

- Run batch jobs: Learn how to run a batch job using the Code Engine console. A job runs one or more instances of your executable code. Unlike applications, which handle HTTP requests, jobs are designed to run one time and exit.

- Serverless web app and eventing for data retrieval and analytics. Create an application to automatically collect GitHub traffic statistics for repositories and provide the foundation for traffic analytics.

- Quick lab: No infrastructure, just code. See the simplicity of serverless: In this 45-minute lab, you'll create an IBM Cloud account and then use Node.js to create an action, an event-based trigger, and a web action.

Serverless and IBM Cloud

Serverless computing offers a simpler, more cost-effective way of building and operating applications in the cloud. And it can help smooth the way as you modernize your applications on your journey to cloud.

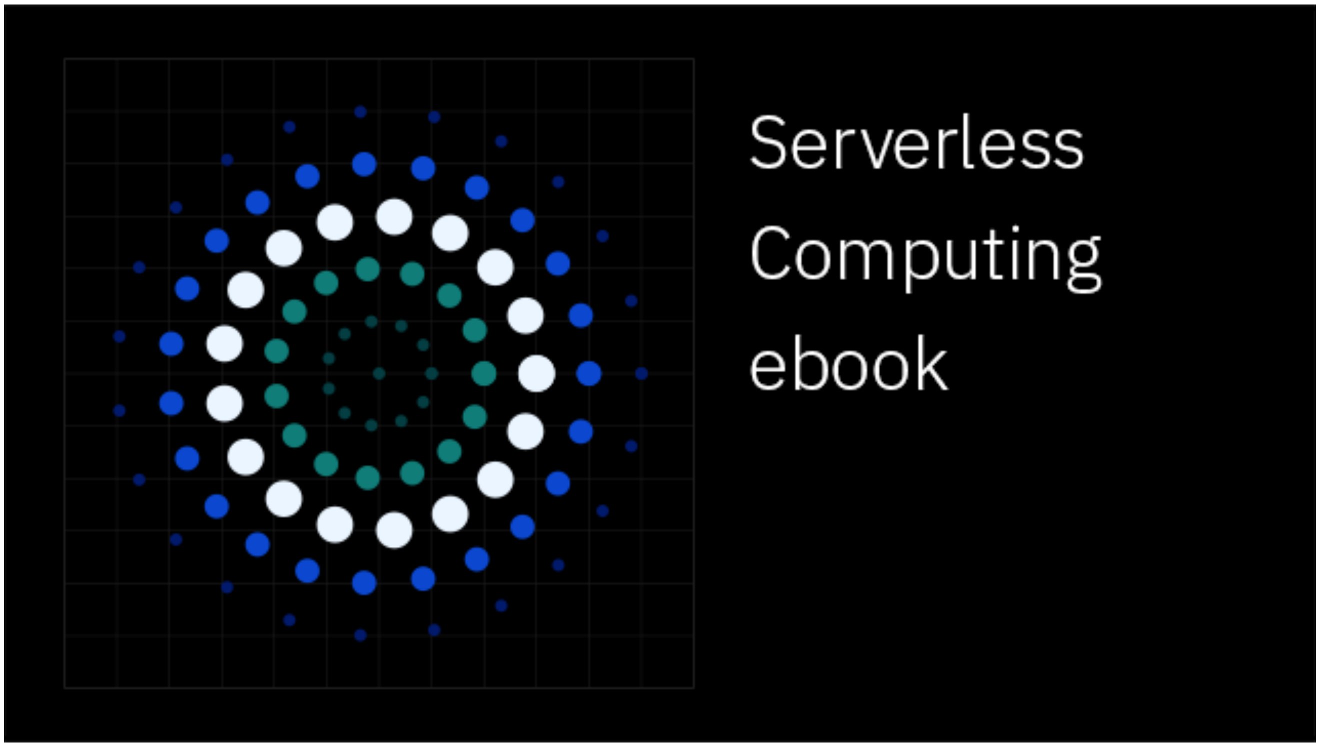
Take the next step:

- Learn about IBM Cloud Code Engine, a pay-as-you-use serverless platform that lets developers deploy serverless applications and workflows combining source code, container images or batch jobs, with no Kubernetes skills needed.

- Learn how you can deploy and run your serverless apps consistently across on-premises data centers, edge computing environments, and any vendor’s public cloud environments using IBM Cloud Satellite.

- Check out other IBM products and tools that can be used together with IBM Cloud Code Engine, including IBM Watson APIs, Cloudant, Object Storage, and Container Registry.

Get started with an IBM Cloud account today.



Serverless in the enterprise, 2021

New research uncovers insights on the real-world opportunities and challenges of serverless computing.

Download the e-book (1.8 MB)