Serverless

Site feedback

By: **IBM Cloud Education**

28 September 2021

Cloud Compute

What is serverless?

Serverless

Serverless is a cloud execution model that enables a simpler, more cost-effective way to build and operate cloud-native applications.

IBM Cloud

Serverless is a cloud computing execution model that

- Automatically provisions the computing resources required to run application code on demand, or in response to a specific event;
- Automatically scales those resources up or down in response to increased or decreased demand;
- Automatically scales resources to zero when the application stops running.

Serverless offloads all management responsibility for backend cloud infrastructure and operations tasks - provisioning, scheduling, scaling, patching and more - to the cloud provider. This gives developers more time to develop and optimize their front-end application code and business logic. And with serverless, customers never pay for idle capacity. They pay only for the resources required to run their applications, and only when those applications are running.

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

Amazon Web Services introduced serverless in 2014 with AWS Lambda; today every leading cloud service provider offers a serverless platform including Microsoft Azure (Azure Functions), Google Cloud (Google Cloud Functions) and IBM Cloud (IBM Cloud Code Engine). Together serverless, microservices and containers, form a triumvirate of technologies typically considered to be at the core of cloud-native application development.

Serverless vs. FaaS (function as a service)

Serverless and function as a service (FaaS) are often conflated. But FaaS is actually a subset of serverless - it's the compute paradigm central to serverless, wherein application code or containers run only in response to events or requests. Serverless includes FaaS plus all the other associated resources and cloud services and resources supporting the code - e.g. storage, databases, networks, API gateways, authentication - for which configuration, management and billing of services are invisible to the user.

What is Serverless?

0:00 / 6:37

Site feedback

Serverless pros and cons

Pros

Serverless offers a number of individual technical and business benefits:

- 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.
- Also as noted above, serverless customers pay for execution only. The meter starts when the request is made, and ends when execution finishes. Compare to the infrastructure as a service (IaaS) compute model, where customers pay for the virtual machines (VMs) and other resources required to run applications, from the time they provision those resources until the time the customer explicitly decommissions those resources.
- Serverless is a polyglot environment, enabling developers to code in any language or framework - Java, Python, node.js - with which they're comfortable.

IBM Cloud

integrate, test, deliver and deploy code builds into production.

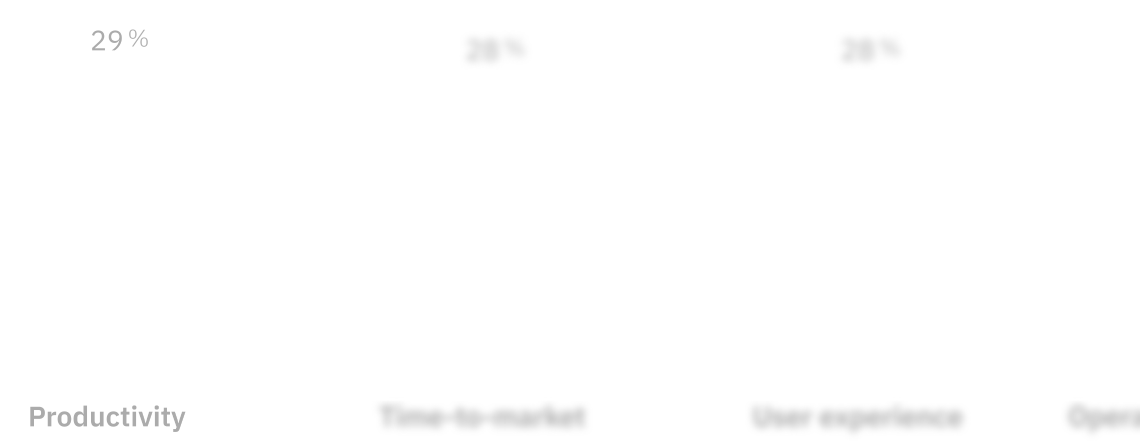
- For certain workloads, such as ones that require parallel processing, serverless can be both faster and more cost-effective than other forms of compute.

- Serverless application development platforms provide near-total visibility into system and user times, and can aggregate that information systematically.

Development and IT professionals cite other specific benefits of serverless computing - 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 computing, organizations are using it for a wide variety of applications (see Figure 2 below). However, there are certain applications for which serverless is not favorable, or which present technical and business trade-offs to consider:

- 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.

- Cold starts: Because serverless architectures forgo long-running 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 certain applications, this startup latency isn't noticeable or detrimental to users. But for others - for example, financial trading application - the delay is unacceptable.

- Monitoring and debugging: These operational tasks are challenging in any distributed system, but a move 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: Serverless architectures are designed to take advantage of an ecosystem of managed cloud services and, in terms of architectural models, go the furthest to decouple a workload from something more portable, like a virtual machine (VM) or Docker container. For some companies, deeply integrating with the native managed services of a specific cloud platform is where much of the value of cloud can be found; for others, this cloud lead to material lock-in risks that need to be mitigated.

Understanding the serverless stack

Defining serverless as a set of common attributes, instead of as an explicit technology, makes it easier to understand how the serverless approach can manifest in other core areas of the stack.

- **Functions as a Service (FaaS):** Again, FaaS is widely understood as the center of most serverless architectures. See "What is FaaS?" for a deeper dive into the technology.

- **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 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.

Comparing FaaS to PaaS, containers, and VMs

While FaaS, Platform as a Service (PaaS), containers, and virtual machines (VMs) all play a critical role in the serverless ecosystem, FaaS is the core compute model for serverless. For that reason, it's useful to explore how FaaS differs from the other compute models available across some key attributes:

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

- **Ongoing administrative burden:** None for FaaS, compared to continuum from light to heavy for PaaS, containers and VMs respectively.

- **Elastic scaling:** Instant and inherent, with auto-scaling to zero, for FaaS; the other models offer automatic but slow scaling that requires careful tuning of auto-scaling rules, and no scaling to zero.

- **Capacity planning:** None required for FaaS; the other models require a mix of some automatic scalability and some capacity planning.

- **Persistent connections and state:** Limited for FaaS, which is inherently stateless; any state must be 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.

- **Maintenance:** Managed 100% by the provider. This is also true for PaaS, but containers and VMs require significant maintenance including updating/managing operating systems, container images, connections, etc.

- **High availability (HA) and disaster recovery (DR):** Inherent in FaaS with no extra effort or 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:** 100% efficient with FaaS because with FaaS there are no such things thing as idle resources—they are invoked only upon request. All other models feature at least some degree of idle capacity.

- **Charging granularity and billing:** Metered in units of 100 milliseconds for FaaS, vs. by the hour (or sometimes by the minute) for the other models.

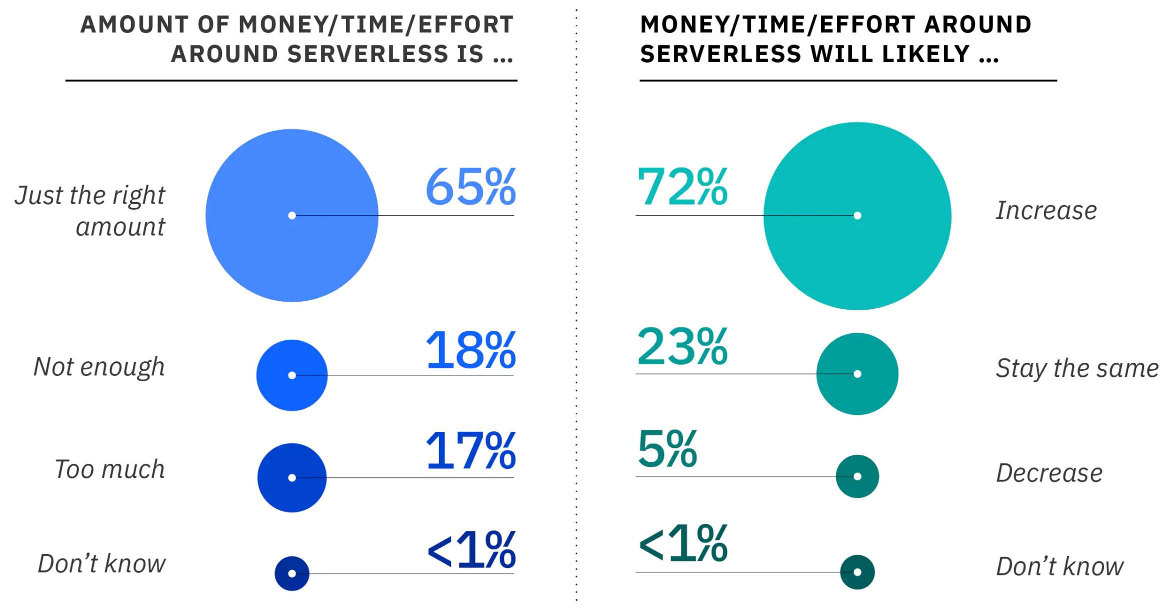


Figure 1: Organizations will continue to invest in serverless. *95 percent of users plan*

Serverless, Kubernetes and Knative

Kubernetes is an open-source container orchestration platform that automates schedules 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.

Enter Knative, 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 other application code (e.g., microservices). It does this 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 it as a serverless function behind the scenes.

Learn more about Knative

Site feedback

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 idl

IBM Cloud

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).

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 (see Figure 2).

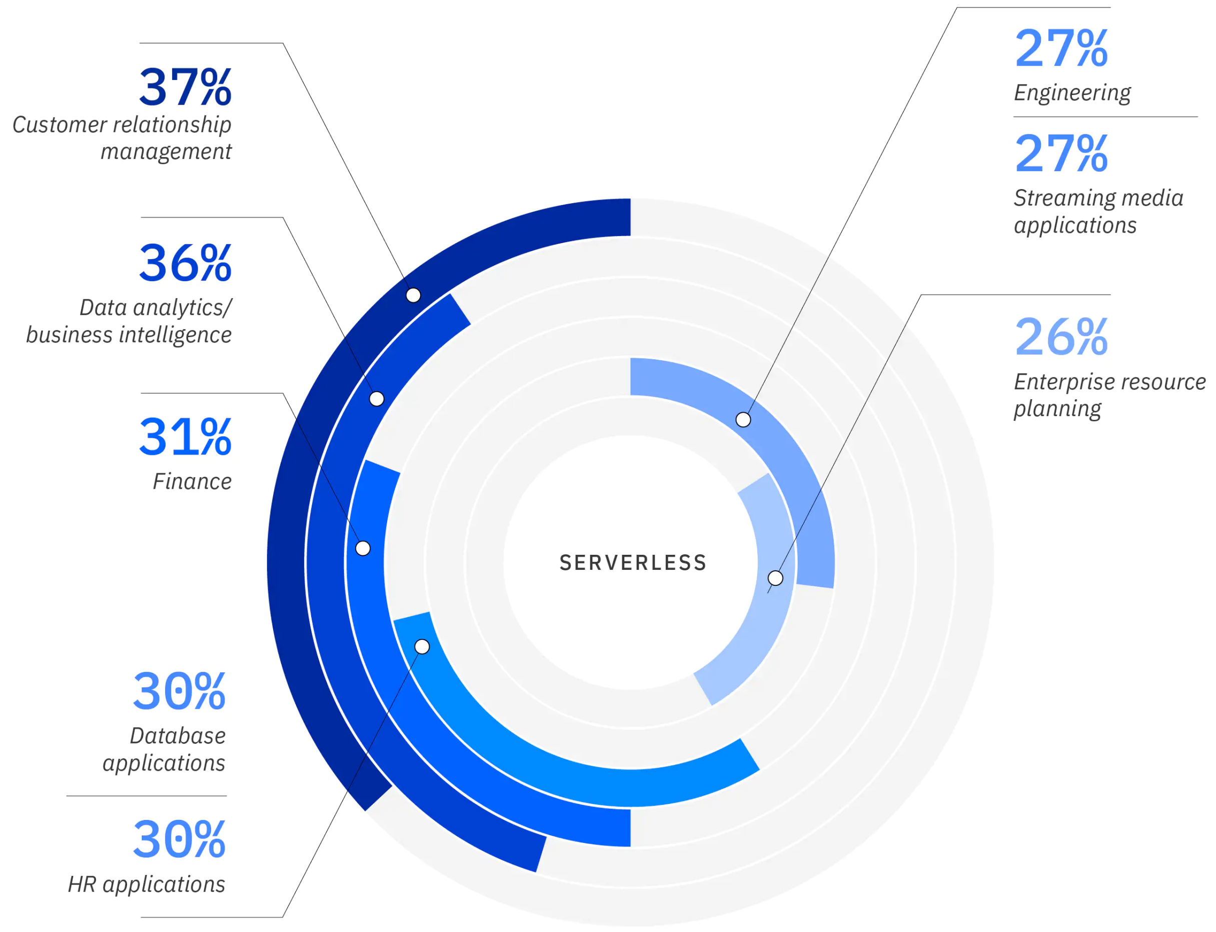


Figure 2: How serverless is being used. *Survey respondents identified over a dozen serverless applications in use. The most commonly cited applications included CRM, data analytics/business intelligence, finance, database, HR, engineering, streaming media and ERP. (Source: Source: 'Serverless in the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

Tutorials: Get started with serverless computing

Expand your serverless computing skills with these tutorials:

- Getting started with IBM Cloud Code Engine: Visit our “Hello world” tutorial to

- 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

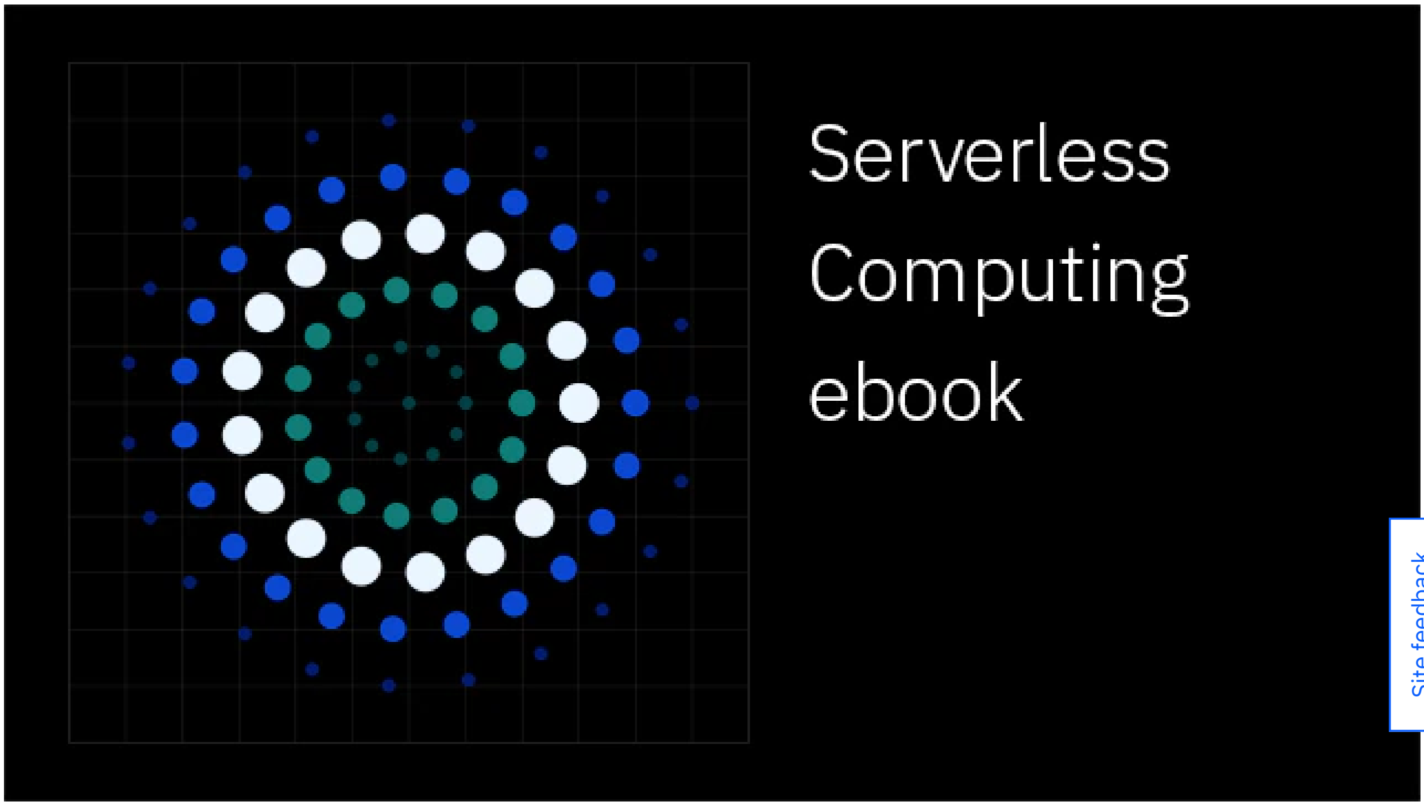
Serverless computing offers a simpler, more cost-effective way of building and operating applications in the cloud. And it can help smooth the way as you modernize your applications on your journey to cloud.

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.



Site feedback

Serverless in the enterprise, 2021

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

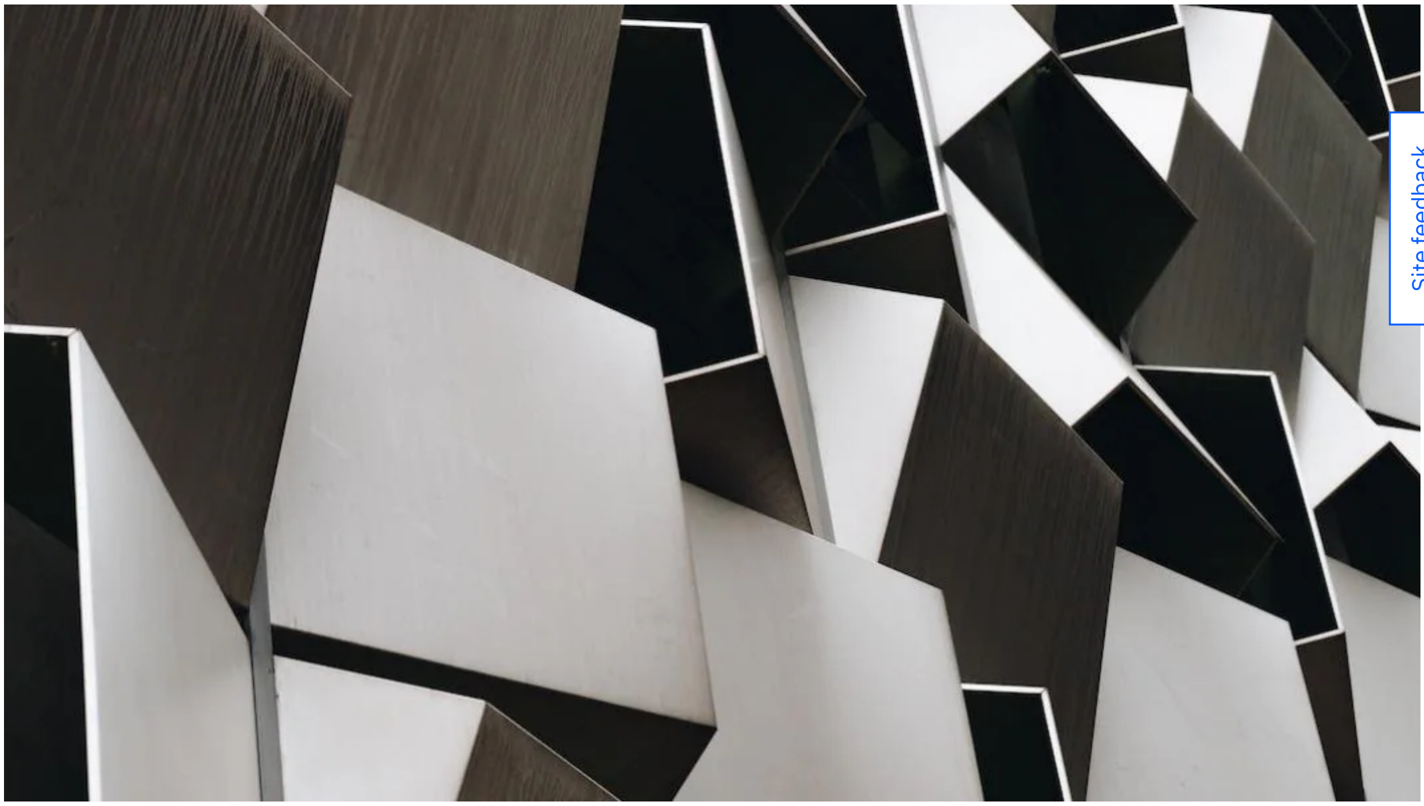
Download the e-book (1.8 MB)

IBM Cloud

Intro to IBM Cloud Code Engine

Go beyond functions to run all your containerized workloads - including web apps, microservices, and batch jobs - on this fully managed serverless platform.

Watch the video (3:59)



Site feedback

Enjoy your cloud again

Take a closer look at IBM Cloud Code Engine and the benefits it offers.

Read the blog →

Featured products

IBM Cloud Code Engine

IBM Cloud

Cloudant

IBM Cloud Object Storage

Container Registry

Why IBM Cloud

Why IBM Cloud

Hybrid Cloud approach

Trust and security

Open Cloud

Data centers

Case studies

Site feedback

Products and Solutions

Cloud Paks

Cloud pricing

View all products

View all solutions

IBM **Cloud**

Learn about

What is Hybrid Cloud?

What is Cloud Computing?

What is Confidential Computing?

What is a Data Lake?

What is a Data Warehouse?

What is Artificial Intelligence (AI)?

What is Machine Learning?

What is DevOps?

What is Microservices?

Site feedback

Resources

Get started

Docs

Architectures

IBM Garage

Training and Certifications

Partners

Cloud blog

Hybrid Cloud careers

My Cloud account

Let’s talk