The American Journal of Interdisciplinary Innovation and Research
ISSN 2642-7478 | Open Access

Type Original Research
PAGE NO. 13-22
DOI 10.37547/tajiir/Volume07Issue10-02

Check for updates

Serverless & Event-Driven Architectures: Redefining Distributed System Design

Swati Karni

Swati Karni. (2025). Serverless & Event-Driven Architectures: Redefining Distributed System Design. The American Journal of Interdisciplinary Innovations and Research, 7(10), 13–22. https://doi.org/10.37547/tajiir/Volume07Issue10-02

OPEN ACCESS

SUBMITTED 14 September 2025
ACCEPTED 26 September 2025
PUBLISHED 06 October 2025
VOLUME Vol.07 Issue 10 2025

CITATION

Abstract- The rise of serverless and event-driven architectures is changing how we design and implement distributed systems. These approaches eliminate the need for server management and use event-based execution. They create systems that are more scalable, easy to deploy, resilient, and cost-effective. This paper looks at how serverless computing shifts the focus from managing infrastructure to focusing on application logic. At the same time, event-driven models improve responsiveness through asynchronous and independent communication. Together, they enhance the design of distributed systems by increasing fault tolerance, lowering operational costs, and supporting real-time, data-heavy applications. The discussion covers architectural principles, design choices, and new best practices that help integrate serverless and event-driven methods in current distributed computing. Ultimately, this study shows how these approaches lead to next-generation cloud-native systems that are flexible, quick, and tailored for changing workloads.

Keywords: Serverless Computing, Event-Driven Architectures, Distributed Systems, Cloud-Native Design, Scalability, Fault Tolerance, Asynchronous Processing, Real-Time Applications, System Resilience, Cloud Architecture

1. Introduction

Designing distributed systems has traditionally been a complex task, often bogged down by the need to manage infrastructure, handle scaling limits, and maintain large, tightly coupled applications. But times are changing. The growth of serverless computing and event-driven architectures is opening new doors for how we build and operate these systems. Rather than stressing over servers and their maintenance, developers can now focus on building useful application logic that truly helps users.

Serverless platforms take care of provisioning, scaling, and managing the underlying hardware, allowing systems to adjust effortlessly to varying workloads. Event-driven models organize applications into separate, independent parts. Each part works on its own and communicates by sending messages called events. This clear separation allows the system to react more promptly to changes, solve problems more effectively, and lets each part evolve or improve without disrupting the others. Today's cloud-native systems use these ideas together to build software that can handle everything from live data streams to complex data analysis, all while keeping costs low and the system easy to manage because events trigger compute functions on demand, resources are used more efficiently — organizations pay only for what they consume, avoiding wasted capacity (Haller et al., 2023).

In this paper, we look at how serverless and event-driven approaches are changing the way distributed systems are designed. We explain the main ideas behind these models, practical design tips, and new best practices. This study shows how these approaches help software teams create systems that are scalable, reliable, and cost-efficient, ready to meet the needs of today's fast-changing application world.

2. Evolution From Bare Metal to Serverless

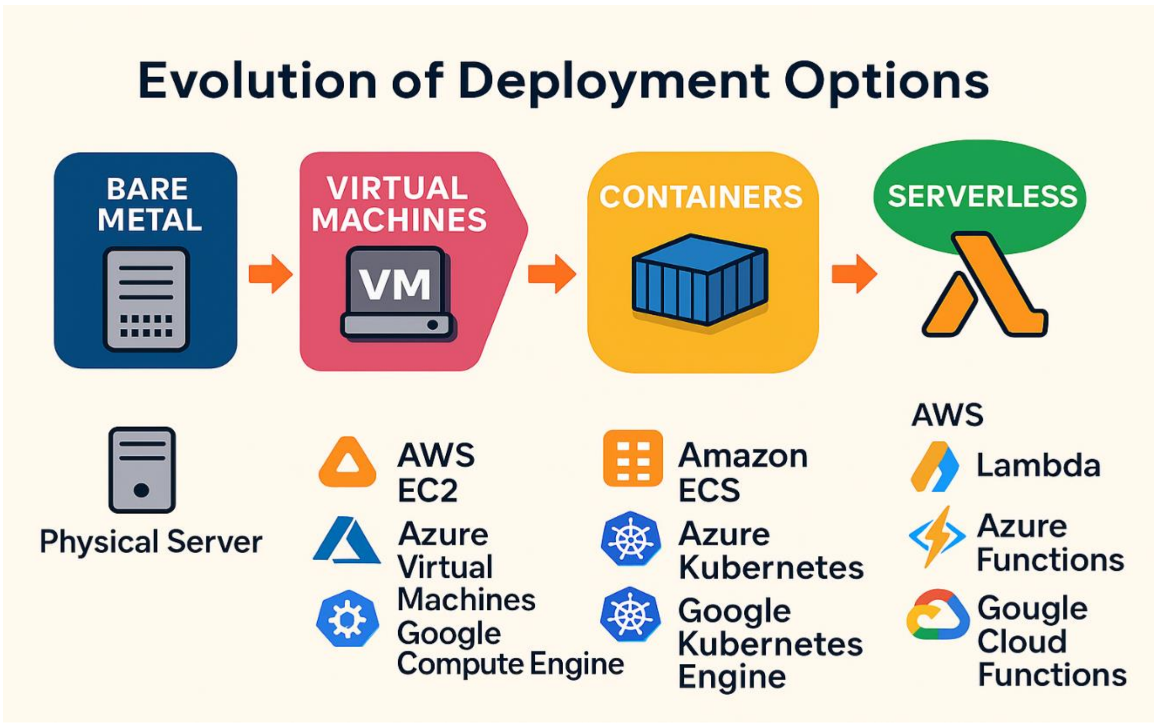


Figure 1: System Deployment Evolution: From Bare Metal to Serverless

Figure 1 shows the changes in deployment over time. It began with Bare Metal, where apps ran on physical servers. Then came the era of Virtual Machines, allowing multiple systems to share a single server. Then Containers made it easier to move and run apps anywhere. Today, Serverless enables developers to simply write code while the cloud takes care of everything else. This change has made deployment faster, easier, and cheaper.

As organizations moved to the cloud, the demand for agility and efficiency pushed providers to offer models that abstracted away even more of the operational expenditure and burden. Serverless computing arose to meet the increasing demand for more efficient development. It builds on utility-based computing by allowing developers to deploy code without having to manage servers. This model fits well with the pay-as-you-go approach of cloud economics, making it appealing to both startups and large companies that want to cut costs while speeding up development. Event-driven architectures are now widely used as modern systems—such as data-intensive apps, IoT devices, and real-time analytics—need to react instantly to new signals. Instead of being tightly linked like traditional systems, they keep components independent and connected through messages. This approach makes systems more scalable and dependable in changing conditions. When combined, serverless and event-driven models draw from older concepts of distributed systems but introduce more automation, flexibility, speed and responsiveness. This demonstrates that they are not just fleeting trends but significant changes in how today's applications are designed and maintained. Together, serverless and event-driven models expand on earlier distributed system ideas while providing greater levels of automation, speed and responsiveness. This shows that these approaches are not just trends but important changes in how modern applications are designed, built, and delivered.

3. Basic Definitions

- **Serverless Computing**

  In serverless computing, the cloud provider handles things like setup, updates, scaling, and maintenance. Developers just write code, and the cloud runs it, charging only for actual use. 'Serverless' means developers don't see or manage servers, even though servers are still there.

  Key Points- Developers don't manage servers or runtime environments. Costs are based on actual compute consumption—no cost for idle time. Code runs in ephemeral, stateless compute containers, spun up and down as needed. Cloud providers handle availability, scaling, and maintenance.

- **Function-as-a-Service (FaaS)**

  Function-as-a-Service (FaaS) is a core approach within serverless computing in which developers write discrete "functions" that execute in response to specific events (such as HTTP requests, file uploads, or database changes). Each function is responsible for a specific, stateless activity and is triggered by defined events—enabling highly modular and scalable applications.

  Key Points- Code is executed as functions, not as continuously running services. Functions are triggered by events and run for short durations.

  Examples- AWS Lambda, Azure Functions, Google Cloud Functions.

- **Event-driven architecture**

  Serverless computing often relies on event-driven architectures. In this model, system components communicate by producing and consuming events. Functions are activated only in response to triggers such as message queues, HTTP calls, scheduled events, database updates, or file storage activity.

- **Backend-as-a-Service (BaaS)**

  Backend-as-a-Service (BaaS) is another serverless pattern, where cloud providers expose backend functionalities (like databases, authentication, notifications, etc.) as managed services. Developers use APIs rather than building and maintaining backend components themselves.

- **Statelessness**

  Statelessness in serverless means that each function invocation is independent, with no persistent memory retained between calls. Any required state must be managed outside the function—typically in a database or storage service.

- **Dynamic Scalability**

  Serverless architectures provide dynamic scalability: resources automatically scale up or down in response to demand without manual intervention. Functions can handle both zero and thousands of concurrent requests seamlessly.

- **Pay-As-You-Go Billing**

  Pay-as-you-go (or pay-per-use) billing is a key property of serverless. Billing is based on the actual duration and resources consumed by each function execution, rather than prepaid or reserved capacity.

- **Vendor Lock-in**

  Because serverless uses cloud-provider-specific runtimes, there is potential for vendor lock-in—difficulty migrating workloads to another platform due to proprietary APIs or services.

- **Microservices**

  Decomposing applications into small, independently deployable services—often used alongside serverless for agile, scalable systems.

  Serverless Containers: Some platforms support container-based serverless deployment, offering more flexibility for complex workloads.

  Edge Computing: Serverless functions can be run at edge locations, closer to users, for lower latency applications.

4. Tools and Methodologies

This section describes the cloud platforms, materials, and methodologies used to explore serverless and event-driven architectures for modern distributed system design.

4.1 Cloud Platforms and Tools

The implementation leverages services from three major cloud providers—**Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP)**—each offering comprehensive serverless and event-driven computing frameworks.

- **AWS Lambda:** A Function-as-a-Service (FaaS) platform that executes code in response to events without server provisioning. Lambda automatically scales and integrates with AWS services like API Gateway, S3, SNS, SQS, and Event Bridge for event routing (Amazon Web Services, 2025).
- **Azure Functions:** Microsoft's equivalent serverless compute service supports multiple hosting plans and integrates with Azure Blob Storage, Cosmos DB, Event Grid, and Service Bus. Azure Function bindings simplify connecting functions to various data sources and event sources (Microsoft Azure, 2025).
- **Google Cloud Functions:** GCP's event-driven serverless compute service that runs functions triggered by HTTP requests, Pub/Sub messages, or Cloud Storage changes. It automatically scales with demand and supports popular runtimes (Google Cloud Platform, 2025).
- **Google Cloud Run:** Google Cloud Run is a fully managed platform that allows deployment of containerized applications without managing servers. It automatically scales based on traffic and only charges for the resources used during execution (Google Cloud Platform, 2025).

4.1.1 Event Routing and Messaging

- **AWS EventBridge**, complemented by SNS and SQS, provides scalable event bus and messaging services.
- **Azure Event Grid** and **Service Bus** offer enterprise event routing and messaging respectively.
- **Google Cloud Pub/Sub** enables scalable, durable messaging between independent services.

4.1.2 Streaming Services

Amazon Kinesis facilitates real-time streaming data processing. **Azure Event Hubs** and **Google Cloud Dataflow** provide similar streaming analytics capabilities.

4.2 Methods

This approach is built on serverless and event-driven principles:

- **Modular Event-Driven Design:** Systems are made of separate parts that send and receive events through the cloud, making them easier to scale and manage.
- **Event Routing and Filtering:** Rules decide where each event goes, sending it to the right function or service like AWS Lambda, Azure Functions, or Google Cloud Functions for faster processing.
- **Stateless, On-Demand Functions:** Functions run only when triggered, don't save state, and scale automatically to handle sudden spikes in work.
- **Integrated Observability:** Tools like AWS CloudWatch, Azure Monitor, and Google Cloud Operations track performance and errors, keeping systems reliable.
- **Schema Management:** Standard event formats are used so data flows smoothly and works consistently across platforms.
- The combination of these cloud services and methodologies facilitates the creation of scalable, resilient, and cost-efficient distributed systems optimized for real-time event-driven workloads.

5. Traditional Distributed Systems to Serverless Architectures with AI and ML Integration

5.1 Traditional Distributed Systems

Traditional distributed systems use servers or virtual machines that are set up and managed manually. IT teams handle tasks like scaling, updates, monitoring, and failovers. While these systems give full control over hardware and networking, they also create many operational challenges. It results in higher fixed costs, reduced flexibility, and longer time to market for new features or updates. Developers often deploy monolithic applications or microservices, keeping the application state in memory or local storage.

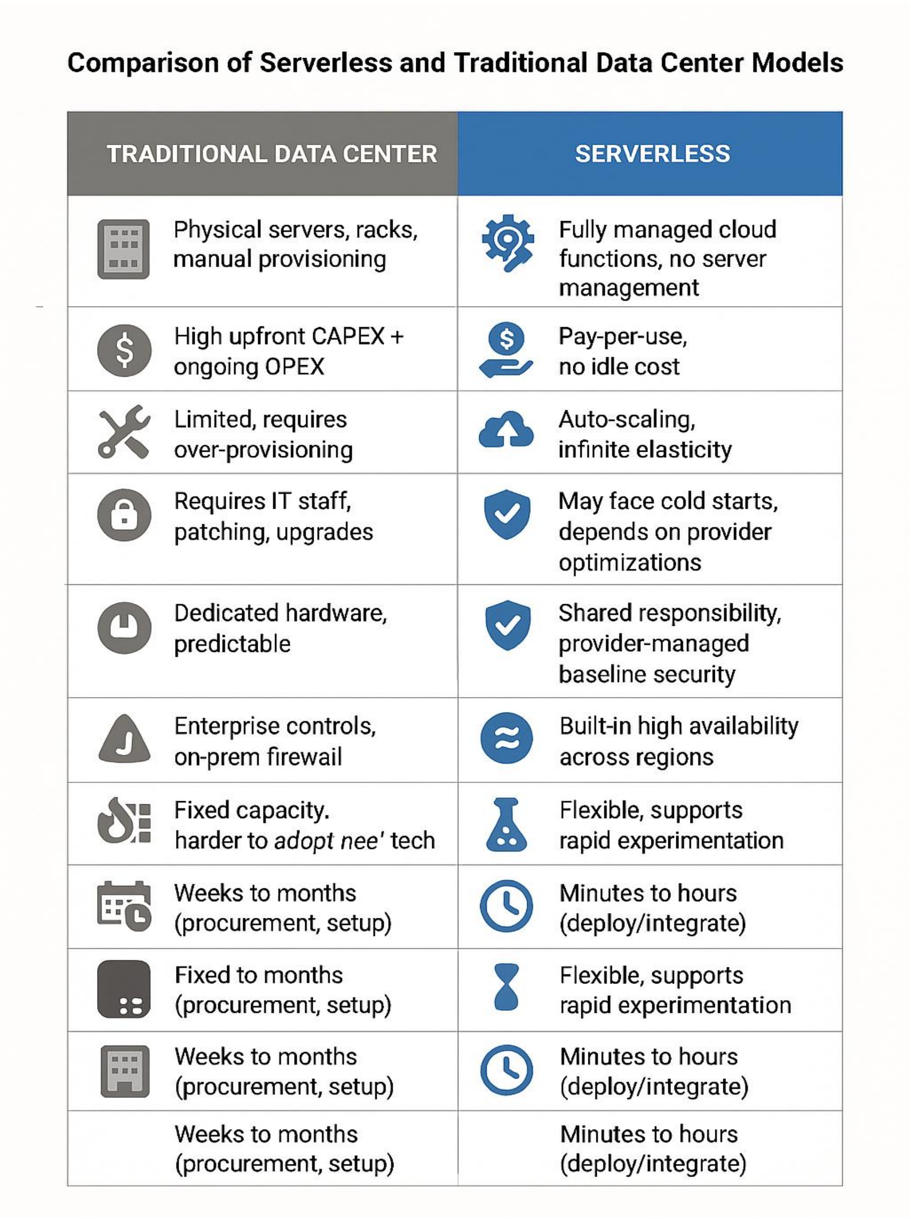
Example: A large e-commerce platform using a cluster of virtual machines for all frontend, backend, and database workloads. IT staff must provision extra capacity in advance for high-traffic events (e.g., Black Friday), resulting in inefficiency and higher costs during off-peak times.

5.2 Serverless Architectures

Serverless computing abstracts away server management entirely. Cloud providers handle all underlying infrastructure, including scaling, patching, and provisioning. Applications are built from stateless, event-driven functions that scale automatically with demand and are billed per execution or resource consumption. Serverless architectures are well-suited for event-driven applications, APIs, and workloads with highly variable traffic.

Key Benefits and Challenges

Serverless systems bring many benefits, such as automatic scaling without extra effort, lower costs, and less work to manage infrastructure. Since applications are broken into smaller, independent functions, developers can also build and update features more quickly. Still, there are some challenges, like having less control over the system, limits on how long each function can run, and possible delays in performance, such as cold starts. Figure 2 shows a comparison of traditional and serverless data center models.



Comparison of Serverless and Traditional Data Center Models

TRADITIONAL DATA CENTER	SERVERLESS
Physical servers, racks, manual provisioning	Fully managed cloud functions, no server management
High upfront CAPEX + ongoing OPEX	Pay-per-use, no idle cost
Limited, requires over-provisioning	Auto-scaling, infinite elasticity
Requires IT staff, patching, upgrades	May face cold starts, depends on provider optimizations
Dedicated hardware, predictable	Shared responsibility, provider-managed baseline security
Enterprise controls, on-prem firewall	Built-in high availability across regions
Fixed capacity. harder to *adopt nee'* tech	Flexible, supports rapid experimentation
Weeks to months (procurement, setup)	Minutes to hours (deploy/integrate)
Fixed to months (procurement, setup)	Flexible, supports rapid experimentation
Weeks to months (procurement, setup)	Minutes to hours (deploy/integrate)
Weeks to months (procurement, setup)	Minutes to hours (deploy/integrate)

Figure 2: Traditional vs Serverless Data Centers

Example: A photo-sharing service processes image uploads using serverless functions: when a new photo is added, an event triggers a Lambda (or Cloud Function/Azure Function) to resize and store the image. Developers do not manage the server lifecycle, and the system only incurs costs when processing images.

5.3 Serverless Architectures with AI and Machine Learning Integration

Recently, serverless platforms have begun to support sophisticated AI and ML workloads. In this paradigm, developers deploy ML models or inference pipelines as serverless functions, enabling real-time decisionmaking, data processing, and intelligent automation at scale. Serverless AI abstracts the complexity of model deployment, making it feasible to update models or scale inference rapidly while only paying for actual execution.

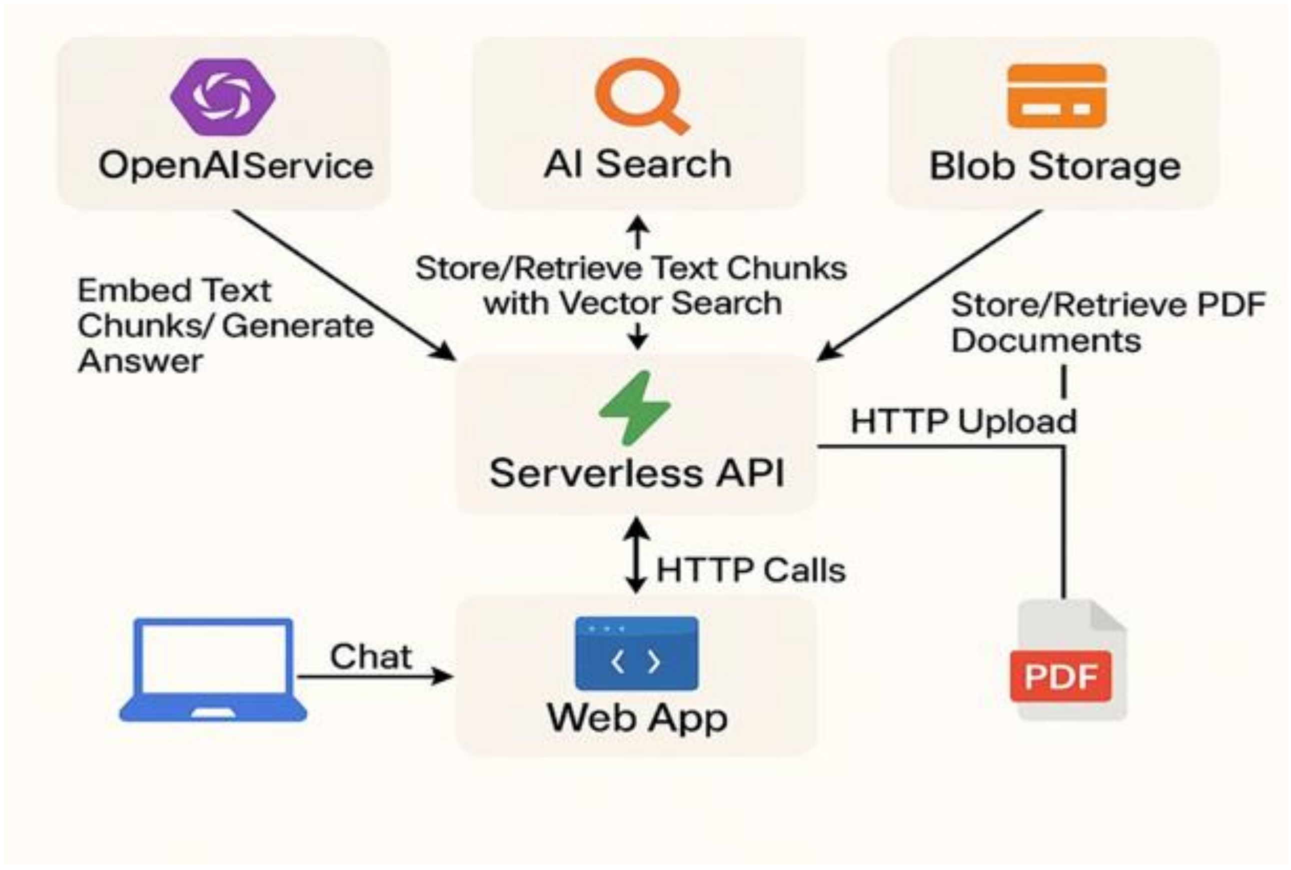

OpenAIService
AI Search
Blob Storage
Embed Text Chunks/ Generate Answer
Store/Retrieve Text Chunks with Vector Search
Store/Retrieve PDF Documents
HTTP Upload
Serverless API
HTTP Calls
Chat
Web App
PDF

Figure 3: Serverless Architecture with AI ML integration (Microsoft Azure, 2025).

Figure 3 shows users interact with a Web App via chat to input queries or upload documents. Uploaded PDF documents are sent via HTTP upload to a Serverless API implemented using Azure Functions. The Serverless API handles multiple tasks, it stores and retrieves PDF documents with Azure Blob Storage, a scalable object storage service. It breaks the contents of the PDFs into text chunks and stores/retrieves them using Azure **AI Search** with vector search capabilities, enabling semantic search over large volumes of data. It sends text chunks to the Azure OpenAI Service for embedding and generating answers, leveraging advanced AI language models to understand and respond to natural language queries. The Serverless API communicates with the Web App through HTTP calls, facilitating chat responses based on processed AI-generated insights.

The architecture is fully serverless, allowing automatic scaling and no management of infrastructure. It integrates AI-powered services including natural language processing (Azure OpenAI Service) and semantic search (Azure AI Search). Designed for document-centric conversational experiences, enabling users to query large document repositories interactively. Modular and event-driven, it enables on-demand function execution triggered by user actions and data events.

Key Benefits and Challenges

Using serverless platforms for AI and ML offers several advantages. It makes it easier and faster to deploy custom or pretrained models, while keeping costs low since resources are only used during requests or inference, not when sitting idle. Cloud providers also handle scaling and scheduling automatically, assigning GPUs, TPUs, or CPUs as needed—whether for HuggingFace models, Google Vertex AI, or AWS SageMaker endpoints.

Despite these benefits, integrating serverless architecture with AI brings unique and complex technical challenges.

- Cold Start Latency: Deploying large AI models involves significant delays due to loading model weights, which can be several GBs, into accelerators and starting execution contexts. This can result in long waits for the first inference ("cold start"), which is less of a problem in always-on traditional setups (SIGARCH, 2025).

- **State Management:** While traditional serverless functions are stateless, many modern AI workloads require state, such as cached model parameters or context for sequential predictions. Handling state efficiently within a serverless paradigm is non-trivial and may impact performance if not handled judiciously (SIGARCH, 2025).
- **Complex Communication Patterns:** AI pipelines often require the chaining of multiple model inferences or rapid data transfer between functions, which is challenging in systems designed assuming independent, stateless functions.
- **Advanced Scheduling Needs:** Serverless providers must make intelligent, real-time decisions about hardware allocation, batch sizes, and parallelism modes to achieve high efficiency. These scheduling problems are compounded by the bursty, stateful, and sometimes unpredictable nature of AI workloads.

Example:

- **Real-time fraud detection:** When a financial transaction occurs, a serverless function invokes an AI model to assess risk and flag suspicious activity instantly. The cloud provider automatically handles scaling, infrastructure, and hardware selection to meet demand as transaction volumes spike.
- **Conversational AI and Chatbots:** Customer messages trigger serverless functions, which run ML-powered natural language processing to generate responses, integrating with platforms like AWS Lambda, Azure Functions, or Google Cloud Functions and AI services (Lex, Bot Service, Dialogflow).

6. Established use cases of serverless

Use Case	Description / Why Serverless Works Well	Examples / Research Evidence
Web & Mobile App Backends / APIs	Many applications need RESTful APIs or similar backends that respond to HTTP requests, scale up/down with demand, and often have irregular or spiky load. Serverless allows you to avoid provisioning infrastructure for low-traffic periods and scale automatically.	DigitalOcean identifies APIs for web & mobile apps as a top use case. (digitalocean.com) (DigitalOcean, 2023)
Event-driven Processing / Triggered Functions	When tasks are initiated by events (e.g. file upload, database change, message queue), serverless functions can automatically react without always-on servers. Works well for workflows where the operations are stateless or can be decomposed.	"The Rise of Serverless Computing" (ACM) describes image thumbnail creation on S3 upload, event handlers, etc. (cacm.acm.org) (Jonas et al., 2019).
Data Processing / ETL / Pipelines	Handling ingestion, transformation, filtering of data (batch & streaming) benefits from serverless since many operations happen when data arrives, not constantly. Enables cost savings and often simpler operational model.	SPEC RG's "A Review of Serverless Use Cases" survey includes many data processing workflows. (research.spec.org); also Lambada project shows interactive data analytics on cold data using serverless architecture. (arXiv) (Eismann et al., 2020; Müller et al., 2019).

Internet of Things (IoT)	IoT devices often produce data infrequently or irregularly, or events triggered by physical sensors. Serverless allows handling these with event triggers, with scaling, without dedicated backend servers.	Survey (Hassan et al.) identifies IoT / mobile applications among FaaS use cases. (SpringerOpen); also general overviews (e.g. Xenonstack) list IoT. (xenonstack.com) (Hassan et al., 2021).
Multimedia Processing (e.g. image/video work)	Functions can be triggered by uploads or media arrival; tasks like transcoding, resizing, thumbnail generation are well suited to serverless because they can run independently and scale when needed.	ACM article's "thumbnail creation" example. (cacm.acm.org); DigitalOcean mentions "multimedia processing" among top use cases. (digitalocean.com) (Jonas et al., 2019).
Chatbots / Voice Assistants	These often receive bursts of traffic and depend on external triggers (user messages, voice input). The stateless short tasks make them well-suited for serverless. Scaling down to zero in idle times helps.	Survey by Hassan et al. includes "chatbot" use case. (SpringerOpen); industry examples via blogs. (xenonstack.com) (Hassan et al., 2021).
Real-time / Near-Real-Time Monitoring and Alerting	For example, reacting to telemetry, sensor data, logs, where events must be processed as they arrive. Serverless can help build pipelines or functions that trigger alerts. Also good for infrastructure or operations monitoring.	SPEC RG survey includes scientific use cases like near-real-time water monitoring (Copernicus Sentinel-1) etc. (research.spec.org) (Eismann et al., 2020).
Parallel & Burst-Parallel Computations (Workflows / DAG Jobs)	For workloads that can be expressed as many small independent tasks (e.g. DAGs), serverless can be more cost-efficient and can scale massively for bursts. Key challenges often include scheduling, data locality, managing cold starts.	"Wukong: A Scalable and Locality-Enhanced Framework" shows serverless parallel computing outperforming alternatives in certain workloads. (arXiv); also Lambada for data analytics. (arXiv) (Carver et al., 2020).
Scientific & Earth-Observation Applications	Big science/data workflows (e.g. satellite data, environmental monitoring) where data arrives in batches or streams, processed for insights; sometimes used by government or research institutions.	SPEC RG survey's scientific use cases: e.g. "Copernicus Sentinel-1 for near-real time water monitoring," "Terra Byte - High Performance Data Analytic for Earth Observation," etc. (research.spec.org) (Eismann et al., 2020).

7. Future Scope

- **Serverless-Enabled AutoML and ML Lifecycle Management:** A future possibility is combining serverless computing with automated machine learning (AutoML) and model pipelines. In this setup, tasks like tuning parameters, searching models, monitoring, retraining, and rolling back can all be done with serverless functions, reducing the need for manual work. This helps organizations deploy AI systems faster and at lower cost. The main difficulties are handling the extra orchestration, limits on execution time, and making results consistent across different functions (Vaibhav, 2023).
- **Adaptive Training Frameworks with Budgets and Deadlines:** A future use of serverless is adaptive ML training, where resources can grow or shrink during training to meet set budgets or deadlines. Unlike fixed clusters, this saves cost and avoids wasted compute, making it useful for short or experimental

jobs. The main challenges are time limits, stateless functions, and the need for better scheduling, saving progress, and managing large training tasks (Li et al., 2022).

- **AI-Driven Solutions for Cold Start Optimization:** One of the biggest issues with serverless is the delay, or “cold start,” when functions are triggered for the first time. A possible future solution is using AI itself to reduce this problem. For example, predictive models and reinforcement learning could guess when a function will run and prepare it in advance. AI can also adjust memory and concurrency to improve both speed and cost. This creates a feedback loop where AI makes serverless platforms better for AI workloads. The challenge is making accurate predictions for workloads that are irregular without wasting resources (Zhang et al., 2023).
- **Serverless + Edge + On-Device AI:** Combining serverless with edge and on-device AI has strong potential for fast and secure applications. In this approach, some tasks run on edge devices while others run in the cloud, with serverless functions managing the process. This reduces network use, lowers delay, and keeps user data safer. It is especially useful for IoT systems where data comes in continuously but is processed only when needed. The main challenge is building hybrid systems that can split tasks smoothly between cloud and devices while working around issues like low memory, unstable connections, and different types of hardware (GeeksforGeeks, 2024)

8. Conclusion

Serverless and event-driven architectures represent a significant shift in how we build distributed systems. They separate the application logic from the infrastructure and let systems react quickly to events. These approaches reduce complexity, lower costs through pay-per-use, and improve scalability by automatically adjusting resources to match demand. At the same time, they developers focus on business logic, instead of wasting time in managing servers or their lifecycle. However, challenges like cold starts, state management, and debugging remain. Fortunately, ongoing research and improvements in orchestration, AI driven optimization, ML and hybrid cloud integration are helping to tackle these issues. In the end, serverless and event-driven architectures reshape the fundamentals of distributed computing. They provide a model for creating resilient, flexible, systems that are future ready that meet the changing needs of digital transformation.

References

1. Amazon Web Services. (2025). *AWS Lambda Developer Guide*. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
2. Carver, B., Zhang, J., Wang, A., Anwar, A., Wu, P., & Cheng, Y. (2020). *Wukong: A scalable and locality-enhanced framework for serverless parallel computing*. arXiv. https://arxiv.org/abs/2010.07268
3. DigitalOcean. (2023). *Top use cases for serverless computing*. DigitalOcean. https://www.digitalocean.com/blog/top-use-cases-for-serverless-computing
4. Eismann, S., Scheuner, J., van Eyk, E., Schwinger, M., Grohmann, J., Herbst, N., Abad, C. L., & Iosup, A. (2020). *A review of serverless use cases and their characteristics*. SPEC RG Cloud Working Group. arXiv. https://arxiv.org/abs/2008.11110
5. Google Cloud Platform. (2025). *Serverless on Google Cloud*. https://cloud.google.com/serverless
6. SIGARCH. (2025). AI Goes Serverless: Are Systems Ready? https://www.sigarch.org/ai-goes-serverless-are-systems-ready/
7. Haller, A., Atkinson, M., & Smith, J. (2023). Serverless computing: What it is, and what it is not? *ACM Computing Surveys*, *56*(5), 1-34. https://doi.org/10.1145/3587249
8. Hassan, H. B., Bahsoon, R., Kazman, R., Koziolek, A., Litoiu, M., Shang, W., & Zhu, L. (2021). Serverless computing: A survey of opportunities, challenges, and applications. *Journal of Cloud Computing: Advances, Systems and Applications, 10*(1), 1–48. https://doi.org/10.1186/s13677-021-00253-7
9. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C., Khandelwal, A., Pu, Q., Shankar, V., Menezes Carreira, J., Krauth, K., Yadwadkar, N., Gonzalez, J. E., Popa, R. A., Stoica, I., & Patterson, D. A. (2019). *Cloud programming simplified: A Berkeley view on serverless computing*. UC Berkeley Technical Report No. UCB/EECS-2019-3. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf
10. Microsoft Azure. (2025). *Azure Functions Documentation*. https://learn.microsoft.com/en-

us/azure/azure-functions/

11. Microsoft Azure. (2025). Baseline Azure AI Foundry Chat Reference Architecture. Microsoft Learn. https://learn.microsoft.com/en-us/azure/architecture/ai-ml/architecture/baseline-azure-ai-foundry-chat

12. Müller, I., Marroquín, R., & Alonso, G. (2019). *Lambada: Interactive data analytics on cold data using serverless cloud infrastructure*. arXiv. https://arxiv.org/abs/1912.00937