# Optimizing Enterprise Economics with Serverless Architectures

*October 2017*

# Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

# Contents


Introduction 1

Understanding Serverless Applications 2

- Serverless Application Use Cases 3
- Is Serverless Always Appropriate? 5

Evaluating a Cloud Vendor's Serverless Platform 6

- The AWS Serverless Platform 9
- AWS Serverless Platform Capabilities 10

Case Studies 13

- Serverless Websites, Web Apps, and Mobile Backends 14
- IoT Backends 15
- Data Processing 15
- Big Data 17
- IT Automation 17
- Additional Use Cases 18

Conclusion 18

Contributors 18

Further Reading 19

Reference Architectures 19

Document Revisions 19

# Abstract

This whitepaper is intended to help Chief Information Officers (CIOs), Chief Technology Officers (CTOs), and senior architects gain insight into serverless architectures and their impact on time to market, team agility, and IT economics. By eliminating idle, underutilized servers at the design level and dramatically simplifying cloud-based software designs, serverless approaches are rapidly changing the IT landscape.

This whitepaper covers the basics of serverless approaches and the AWS serverless portfolio, and includes a number of case studies that illustrate how existing companies are already gaining significant agility and economic benefits from adopting serverless approaches. This paper illustrates how organizations of all sizes can use serverless architectures to architect reactive, event-based systems and quickly deliver cloud-native microservices at a fraction of conventional costs.

# Introduction

Many companies are already gaining benefits from running applications in the public cloud, including cost savings from pay-as-you-go billing and improved agility through the use of on-demand IT resources. Multiple studies across application types and industries have demonstrated that migrating existing application architectures to the cloud lowers total cost of ownership (TCO) and improves time to market.[1]

Relative to on-premises and private cloud solutions, the public cloud makes it significantly simpler to build, deploy, and manage fleets of servers and the applications that run on them. However, companies today have additional options beyond classic server or VM-based architectures to take advantage of the public cloud. Although the cloud eliminates the need for companies to purchase and maintain their own hardware, *any* server-based architecture still requires them to architect for scalability and reliability. Plus, companies need to own the challenges of patching and deploying to those server fleets as their applications evolve. Moreover, they must scale their server fleets to account for peak load and then attempt to scale them down when and where possible to lower costs—all while protecting the experience of end users and the integrity of internal systems. Idle, underutilized servers prove to be costly and wasteful. Analysts estimate that as many as 85 percent of servers in practice have underutilized capacity.[2]

Serverless compute services like AWS Lambda are designed to address these challenges by offering companies a different way of approaching application design, one with inherently lower costs and faster time to market. *AWS Lambda eliminates the complexity of dealing with servers at all levels of the technology stack, and introduces a pay-per-request billing model where there are no more costs from idle compute capacity*. Additionally, Lambda functions enable organizations to easily adopt microservices architectures. Eliminating infrastructure and moving to a Lambda model offers dual economic advantages:

- Problems like idle servers simply cease to exist, along with their economic consequences. A serverless compute service like AWS Lambda is never “cold” because charges only accrue when useful work is being performed, with millisecond-level billing granularity.

- Fleet management (including the security patching, deployments, and monitoring of servers) is no longer necessary. This means that it isn't necessary to maintain the associated tools, processes, and on-call rotations required to support 24x7 server fleet uptime. Using Lambda to build microservices helps organizations be more agile. Without the burden of server management, companies can direct their scarce IT resources to what matters—their business.

With greatly reduced infrastructure costs, more agile and focused teams, and faster time to market, companies that have already adopted serverless approaches are gaining a key advantage over their competitors.

# Understanding Serverless Applications

The advantage of the serverless approach cited above is appealing, but what are the considerations for practical implementation? What separates a serverless app from its conventional server-based counterpart?

Serverless apps are architected such that developers can focus on their core competency—writing the actual business logic. Many of the app's boilerplate components, such as web servers, and all of the undifferentiated heavy lifting, such as software to handle reliability and scaling, are completely abstracted away from the developer. What's left is a clean, functional approach where the business logic is triggered only when required: a mobile user sending a message, an image uploaded to the cloud, records arriving in a stream, and so forth. An asynchronous, event-based approach to application design—while not required—is very common in serverless applications, because it dovetails perfectly with the concept of code that runs (and incurs cost) only when there is work to be done.

A serverless application runs in the public cloud, on a service such as AWS Lambda, which takes care of receiving events or client invocations and then instantiates and runs the code. This model offers a number of advantages compared with conventional server-based application design:

- There is no need to provision, deploy, update, monitor, or otherwise manage servers. All of the actual hardware and server software is handled by the cloud provider.

- The application scales automatically, triggered by its actual use. This is inherently different from conventional applications, which require a receiver fleet and explicit capacity management to scale to peak load.
- In addition to scaling, availability and fault tolerance are built in. No coding, configuration, or management is required to gain the benefit of these capabilities.
- There are no charges for idle capacity. There is no need (and in fact, no ability) to pre-provision or over-provision capacity. Instead, billing is pay-per-request and based on the duration it takes for code to run.

## Serverless Application Use Cases

The serverless application model is generic, and applies to almost any type of application from a startup's web app to a Fortune 100 company's stock trade analysis platform. Here are a few examples:

- **Web apps and websites** – Eliminating servers makes it possible to create web apps that cost almost nothing when there is no traffic, while simultaneously scaling to handle peak loads, even unexpected ones.
- **Mobile backends** – Serverless mobile backends offer a way for developers who focus on client development to easily create secure, highly available, and perfectly scaled backends without becoming experts in distributed systems design.
- **Media and log processing** – Serverless approaches offer natural parallelism, making it simpler to process compute-heavy workloads without the complexity of building multithreaded systems or manually scaling compute fleets.
- **IT automation** – Serverless functions can be attached to alarms and monitors to provide customization when required. Cron jobs and other IT infrastructure requirements are made substantially simpler to implement by removing the requirement to own and maintain servers for their use, especially when these jobs and requirements are infrequent or variable in nature.
- **IoT backends** – The ability to bring any code, including native libraries, simplifies the process of creating cloud-based systems that can implement device-specific algorithms.

- **Chatbots (including voice-enabled assistants) and other webhook-based systems** – Serverless approaches are a perfect fit for any webhook-based system, like a chatbot. Their ability to perform actions (like running code) only when needed (such as when a user requests information from a chatbot) makes them a straightforward and typically lower-cost approach for these architectures. For example, the majority of Alexa Skills for Amazon Echo are implemented using AWS Lambda.
- **Clickstream and other near real-time streaming data processes** – Serverless solutions offer the flexibility to scale up and down with the flow of data, enabling them to match throughput requirements without the complexity of building a scalable compute system for each application. When paired with a technology like Amazon Kinesis, AWS Lambda can offer high-speed processing of records for clickstream analysis, NoSQL data triggers, stock trade information, and more.

In addition to the highly adopted use cases discussed earlier, companies are also applying serverless approaches to the following domains:

- Big data, such as map-reduce problems, high speed video transcoding, stock trade analysis, and compute-intensive Monte Carlo simulations for loan applications. Developers have discovered that it's much easier to parallelize with a serverless approach,[3] especially when triggered through events, leading them to increasingly apply serverless techniques to a wide range of big data problems without the need for infrastructure management.
- Low latency, custom processing for web applications and assets delivered through content delivery networks. By moving serverless event handing to the edge of the internet, developers can take advantage of lower latency, and the ability to customize retrievals and content fetches easily. This enables a new spectrum of use cases that are latency-optimized based on the client's location.
- Connected devices, enabling serverless capabilities such as AWS Lambda functions to run inside commercial, residential, and hand-held Internet of Things (IoT) devices. Serverless solutions such as Lambda functions offer a natural abstraction from the underlying physical (and even virtual) hardware, enabling them to more easily transition from the data

center to the edge, and from one hardware architecture to another, without disrupting the programming model.

- Custom logic and data handling in on-premises appliances such as AWS Snowball Edge. Because they decouple business logic from the details of the execution environment, serverless applications can easily function in a wide variety of environments, including on an appliance.

Typically, serverless applications are built using a *microservices architecture* in which an application is separated into independent components that perform discrete jobs. These components, which are made up of individual Lambda functions along with APIs, message queues, database, and other components, can be independently deployed, tested, and scaled. In fact, serverless applications are a natural fit for microservices because of their function-based model. By avoiding monolithic designs and architectures, organizations can become more agile because developers can deploy incrementally and replace or upgrade individual components, such as the database tier, if needed.

In many cases, simply isolating the business logic of an application is all that's required to convert it into a serverless app. Services like AWS Lambda support popular programming languages and enable the use of custom libraries. Long-running tasks are expressed as workflows composed of individual functions that operate within reasonable time frames, which enables the system to restart or parallelize individual units of computation as required.

## Is Serverless Always Appropriate?

Almost all modern applications can be modified to run successfully, and in most cases in a more economical and scalable fashion, on a serverless platform. However, there are a few times when serverless is not the best choice:

- When the goal is explicitly to avoid making any changes to an application.
- When fine-grained control over the environment is required, such as specifying particular operating system patches or accessing low-level networking operations, in order for the code to run properly.
- When an on-premises application hasn't been migrated to the public cloud.

# Evaluating a Cloud Vendor's Serverless Platform

When architecting a serverless application, companies and organizations need to consider more than the serverless compute functionality that executes the app's code. Complete serverless apps require a broad array of services, tools, and capabilities spanning storage, messaging, diagnostics, and more. An incomplete or fragmented serverless portfolio from a cloud vendor can be problematic for serverless developers, who might have to return to server-based architectures if they can't successfully code at a consistent level of abstraction.

A serverless platform consists of the set of services that comprise the serverless app, such compute and storage components, as well as the tools needed to author, build, deploy, and diagnose serverless apps. Running a serverless application in production requires a reliable, flexible, and trustworthy platform that can handle the demands of small startups to global, world-wide corporations. The platform must scale *all* of an application's elements and provide end-to-end reliability. Just as with conventional apps, helping developers succeed in creating and delivering serverless solutions is a multi-dimensional challenge. To meet the needs of large-scale enterprises across a variety of industries, a serverless platform should offer the capabilities in the following illustration.

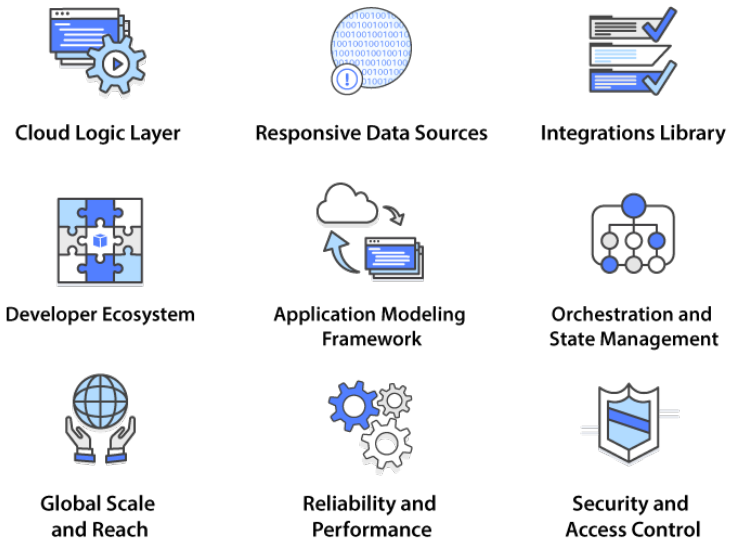



**Figure 1: Capabilities of a serverless platform**

- A high-performance, scalable, and reliable *cloud logic layer*.
- Responsive first-party *event and data sources* and simple *connectivity to third-party systems*.
- *Integration libraries* that enable developers to get started easily, and to add new patterns quickly and safely to existing solutions.
- A vibrant *developer ecosystem* that helps developers discover and apply solutions in a variety of domains, and for a wide set of third-party systems and use cases.
- A collection of fit-for-purpose *application modeling frameworks*.
- *Orchestration* offering state and workflow management.
- *Global scale* and broad reach that includes assurance program certification.
- *Built-in reliability and at-scale performance*, without the need to provision capacity at any level of scale.
- *Built-in security* along with *flexible access control* for both first-party and third-party resources and services.

At the core of any serverless platform is the *cloud logic layer* responsible for running the functions that represent business logic. Because these functions are often executed in response to events, simple *integration with both first-party and third-party event sources* is essential to making solutions simple to express and enabling them to scale automatically in response to varying workloads. For example, serverless functions may need to execute each time an object is created in an object store or for each update made to a serverless NoSQL database. Serverless architectures eliminate all of the scaling and management code typically required to integrate such systems, shifting that operational burden to the cloud vendor.

Developing successfully on a serverless platform requires that a company be able to get started easily, including finding ready-made templates for common use cases, whether they involve first-party or third-party services. These *integration libraries* are essential to convey successful patterns—such as processing streams of records or implementing webhooks—especially during the period when developers are migrating from server-based to serverless architectures. A closely related need is a *broad and diverse ecosystem* that

surrounds the core platform. A large, vibrant ecosystem helps developers readily discover and use solutions from the community and makes it easy to contribute new ideas and approaches. Given the variety of toolchains in use for application lifecycle management, a healthy ecosystem is also necessary to ensure that every language, IDE, and enterprise build technology has the runtimes, plugins, and open source solutions necessary to integrate the building and deploying of serverless apps into existing approaches. It's also critical for serverless apps to leverage existing investments, including developers' knowledge of frameworks such as Express and Flask and popular programming languages. A broad ecosystem provides important acceleration across domains and enables developers to repurpose existing code more readily in a serverless architecture.

*Application modeling frameworks*, such as the open specification AWS Serverless Application Model (AWS SAM) enable a developer to express the components that make up a serverless app, and enable the tools and workflows required to build, deploy, and monitor those applications. Another framework that is key to the success of a serverless platform is *orchestration and state management*. The largely stateless nature of serverless computing requires a complementary mechanism for enabling long-running workflows. Orchestration solutions enable developers to coordinate the multiple, related application components that are typical in a serverless application, while still enabling those applications to be composed of small and short-lived functions. Orchestration services also simplify error handling and provide integration with legacy systems and workflows, including those that run for longer than serverless functions themselves generally permit.

To support customers worldwide, including multinational corporations with global reach, a serverless platform must offer *global scale*, including data centers and edge locations located worldwide. Edge locations are key to bringing low-latency serverless computing close to end users. Because the platform, rather than the application developer, is responsible for supplying the scalability and high availability of serverless apps, its intrinsic *reliability* is critical. Features like built-in retries and dead-letter queues for unprocessed events help developers to construct robust systems with end-to-end reliability using serverless approaches. Performance is equally key, especially low latency (overhead), given that language runtimes and customer code are instantiated on demand in a serverless app.

Finally, the platform must have a broad array of *security and access controls*, including support for virtual private networks, role-based and access-based permissions, robust integration with API-based authentication and access control mechanisms (including third-party and legacy systems), and support for encrypting application elements, such as environment variable settings. Serverless systems, by their design, offer an inherently higher level of security and control for the following reasons:

- **First-class fleet management, including security patching** – In a system like AWS Lambda, the servers that execute requests are constantly monitored, cycled, and security scanned. They can be patched within hours of key security update availability, as opposed to many enterprise compute fleets that can have much looser SLAs for patching and updating.
- **Limited server lifetimes** – Every machine that executes customer code in AWS Lambda is cycled multiple times per day, limiting its exposure to attack and ensuring constantly up-to-date operating system and security patching.
- **Per-request authentication, access control, and auditing** – Every compute request executed on AWS Lambda, regardless of its source, is individually authenticated, authorized to access specified resources, and fully audited. Requests arriving from outside of AWS data centers via Amazon API Gateway provide additional internet-facing defense systems, including DoS attack defenses. Companies migrating to serverless architectures can use AWS CloudTrail to gain detailed insight into which users are accessing which systems with what privileges, and they can use AWS Lambda to process the audit records programmatically.

## The AWS Serverless Platform

Since the introduction of Lambda in 2014, AWS has created a complete serverless platform. It has a broad collection of fully managed services that enable organizations to create serverless apps that can integrate seamlessly with other AWS services and third-party services. Figure 2 illustrates a subset of the components in the AWS serverless platform and their relationships.

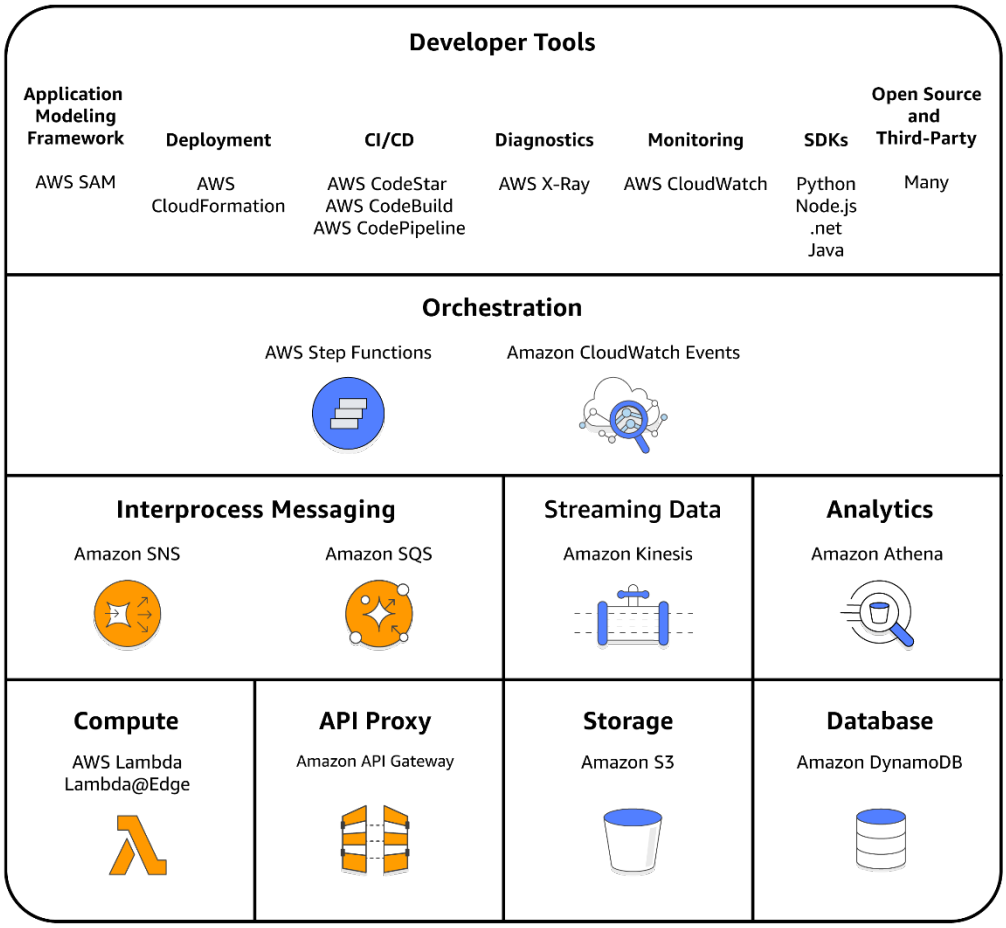


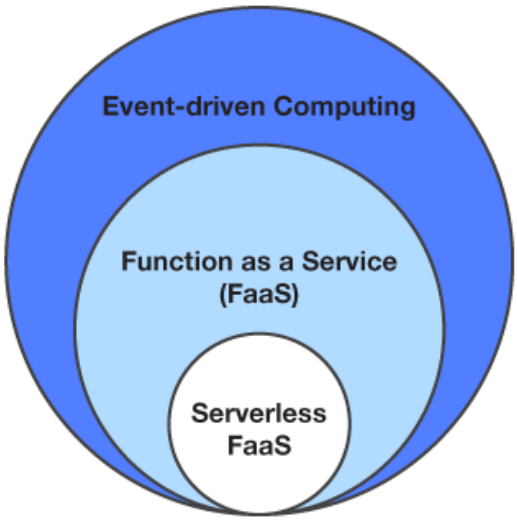

**Figure 2: AWS serverless platform components**

## AWS Serverless Platform Capabilities

AWS provides all the core capabilities identified in the previous section as requirements for a complete serverless platform. The cloud logic layer is provided by AWS Lambda, a high-scale, provision-free serverless compute offering based on functions. AWS Lambda is complemented by AWS Lambda@Edge, which provides similar support for executing extremely low-latency Lambda functions using edge-optimized routing, and AWS Greengrass, which enables Lambda functions to execute on connected devices, including appliances such as AWS Snowball.

Lambda functions can be easily triggered by a variety of first-party and third-party events, enabling developers to build reactive, event-driven systems (see Figure 3) without the conventional hassle of setting up and managing infrastructure. When there are multiple, simultaneous events, Lambda simply

runs more copies of the function in parallel, responding to each individual trigger. Lambda functions scale precisely with the size of the workload, down to the individual request. As a result, there is no possibility of an idle server or container. The problem of wasted infrastructure expenditures is eliminated *by design* in architectures that use Lambda functions.




FaaS, or *Function as a Service*, is one approach to building event-driven computing systems that relies on functions as the unit of deployment and execution. *Serverless FaaS* is a type of FaaS where no virtual machines or containers are present in the programming model and where the vendor provides provision-free scalability and built-in reliability.

**Figure 3: The relationship between event-driven computing, FaaS, and Serverless**

The AWS serverless compute capabilities provided by Lambda are a key element of the following managed services offered by AWS, all of which integrate seamlessly with one another:

- Amazon API Gateway – HTTP endpoints for Lambda functions, including a full range of API proxy and API management capabilities.
- Amazon S3 – Lambda functions can be used as automatic event triggers when an object is created, copied, or deleted.
- Amazon DynamoDB – Lambda functions can be used to process any or all of the changes made to a database table.
- Amazon SNS – Messages can be routed to Lambda functions for processing, adding the ability to dynamically respond to published content.

- Amazon SQS – Messages in queues can be easily processed by Lambda functions.
- Amazon Kinesis Streams – In-order record processing of streaming data is provided by Lambda functions, making it easy to build near real-time analytics engines.
- Amazon Kinesis Firehose – Lambda functions can be applied automatically to the records ingested by a Firehose, making it easy to add transformation, filtering, and analysis capabilities to a data stream.
- Amazon Athena –Lambda functions can be automatically triggered for each object in a query's result set.
- AWS Step Functions – Multiple Lambda functions can be orchestrated to create long-running workflows for both human-centric and automated processes.
- Amazon CloudWatch Events – Lambda functions can be used to automatically respond to events, including third-party events.
- Amazon Aurora – Database triggers can be written as Lambda functions.

Lambda provides an integration library with blueprints for a wide variety of third-party services, including Slack, Algorithmia, Twilio, Loggly, Splunk, SumoLogic, Box, and others, enabling developers to build responsive applications that include analytics, advanced algorithms, communications, and more with just a few lines of code. A wide variety of web application frameworks, including Express (for NodeJS applications) and Flask (for Python applications) have been enhanced to work well with Lambda functions. Open source web application projects include the Serverless Framework, Sparta, Chalice, and many others.

Serverless apps are typically composed of several pieces: one or more functions, a serverless database such as Amazon DynamoDB, and either an API for clients to call or an event source that triggers the app. To keep these pieces organized, AWS uses SAM, the open specification Serverless Application Model. With SAM, developers can easily describe the functions, APIs, event sources, database tables, and other parts of a serverless app. Using SAM also helps developers manage all of the steps in the software development lifecycle, and AWS offers a range of tools and services to help. This includes native support for local testing and debugging in their IDE of choice (or via command line) via

SAM Local, deploying SAM apps using AWS CloudFormation, support for building SAM apps in AWS CodeBuild, and support for GitHub-based CI/CD for SAM apps built into AWS CodePipeline. In addition to first-party support, a number of open source frameworks, CI/CD providers, and performance management vendors offer support for SAM and Lambda functions, including Serverless Framework, Claudia, CloudBees, Datadog, and many more. For additional examples, see Serverless Application Developer Tooling.[4]

After a Lambda function (or in the case of a SAM app, potentially several Lambda functions operating together) has been created, developers can easily monitor it using automatically created metrics and logs available in Amazon CloudWatch and CloudWatch Logs. AWS also offers AWS X-Ray, a cross-service request tracing and performance analysis solution that enables developers to trace the operation and behavior of individual functions and the events that they process.

AWS serverless platform offerings have a global reach, with support for AWS Lambda and Amazon API Gateway in virtually all of the AWS worldwide Regions. Lambda@Edge is available in all edge locations.[5] Lambda offers a range of features to help customers improve the reliability of their applications, including automatic retries for asynchronous and ordered events and dead-letter queues to capture events that were not processed successfully by the application. The deep integration with Amazon Virtual Private Cloud (Amazon VPC) and the flexible range of authentication and access control capabilities provided by AWS Lambda enable organizations to create secure applications that adhere to best practices, such as the principle of least privilege. End user security and management is equally easy – Amazon Cognito offers authorization and authentication that can be easily combined with Amazon API Gateway and AWS Lambda, enabling serverless user registration and sign-in capabilities, including integration with social providers like Facebook and corporate directories.

# Case Studies

Companies have applied serverless architectures to use cases from stock trade validation to e-commerce website construction to natural language processing. AWS Lambda and the rest of the AWS serverless portfolio offer the flexibility to create a wide array of applications, including those requiring assurance programs such as PCI or HIPAA compliance. The following sections illustrate

some of the most common use cases, but are not a comprehensive list. For a complete list of customer references and use case documentation, see Serverless Computing.[6]

## Serverless Websites, Web Apps, and Mobile Backends

Serverless approaches are ideal for applications where load can vary dynamically. Using a serverless approach means no compute costs are incurred when there is no end user traffic, while still offering instant scale to meet high demand, such as a flash sale on an e-commerce site or a social media mention that drives a sudden wave of traffic. Compared to traditional infrastructure approaches, it is also often significantly less expensive to develop, deliver, and operate a web or mobile backend when it has been architected in a serverless fashion.

AWS provides the services developers need to rapidly construct these applications:

- Amazon S3 offers a simple hosting solution for static content.
- AWS Lambda, in conjunction with Amazon API Gateway, provides support for dynamic API requests using functions.
- Amazon DynamoDB offers a simple storage solution for session and per-user state.
- Amazon Cognito provides an easy way to handle end-user registration, authentication, and access control to resources.
- AWS SAM can be used by developers to describe the various elements of an application.
- AWS CodeStar can set up a CI/CD toolchain with just a few clicks.

To learn more, see the whitepaper AWS Serverless Multi-Tier Architectures, which provides a detailed examination of patterns for building serverless web applications.[7] For complete reference architectures, see Serverless Reference Architecture for creating a Web Application[8] and Serverless Reference Architecture for creating a Mobile Backend[9] on GitHub.

### Customer Example – Bustle.com

Bustle.com is a news, entertainment, lifestyle, and fashion website catering to women. It experienced approximately 84 percent cost savings by moving to a serverless architecture based on AWS Lambda and Amazon API Gateway. Bustle's engineers gained additional agility, enabling them to focus on building new product features instead of dealing with infrastructure management and scaling. Bustle's team is now more efficient, using half the people normally required to build and operate sites of Bustle's scale. Bustle's serverless backend also supports the iOS apps for two of its web properties (Bustle and Romper). For more information, see the Bustle case study.[10]

## IoT Backends

The benefits that a serverless architecture brings to web and mobile apps also makes it easy to construct IoT backends and device-based analytic processing systems that seamlessly scale with the number of devices. For an example reference architecture, see Serverless Reference Architecture for creating an IoT Backend on GitHub.[11]

### Customer Example – iRobot

iRobot, which makes robots such as the Roomba cleaning robot, uses AWS Lambda in conjunction with the AWS IoT service to create a serverless backend for its IoT platform. By using a serverless architecture, iRobot's engineering team doesn't have to worry about managing infrastructure or manually writing code to handle availability and scaling. This enables them to innovate faster and stay focused on customers. For more information, see the slides for their AWS re:Invent 2016 presentation Serverless IoT Back Ends (IOT401)[12] or watch the video.[13]

## Data Processing

The largest serverless applications process massive volumes of data, much of it in real time. Typical serverless data processing architectures use a combination of Amazon Kinesis and AWS Lambda to process streaming data, or they combine Amazon S3 and AWS Lambda to trigger computation in response to object creation or update events. When workloads require more complex orchestration than a simple trigger, developers can use AWS Step Functions to create stateful or long-running workflows that invoke one or more Lambda

functions as they progress. To learn more about serverless data processing architectures, see the following on GitHub:

- Serverless Reference Architecture for Real-time Stream Processing[14]
- Serverless Reference Architecture for Real-time File Processing[15]
- Image Recognition and Processing Backend reference architecture[16]

## Customer Example – FINRA

The Financial Industry Regulatory Authority (FINRA) used AWS Lambda to build a serverless data processing solution that enables them to perform half a trillion data validations on 37 billion stock market events daily. In his talk at AWS re:Invent 2016 entitled The State of Serverless Computing (SVR311),[17] Tim Griesbach, Senior Director at FINRA, said "We found that Lambda was going to provide us the best solution for this serverless cloud solution. With Lambda, the system was faster, cheaper, and more scalable. So at the end of the day, we've reduced our costs by over 50 percent and we can track it daily, even hourly."

## Customer Example – Thomson Reuters

Thomson Reuters, a media and information firm, built a serverless business analytics solution that enables their product teams to easily analyze product usage data. The solution combines AWS Lambda, Amazon Kinesis Streams, and Amazon Kinesis Firehose to collect and process streaming event data for analysis. The result, called Product Insight, launched two months ahead of schedule and has exceeded technical expectations.

Anders Fritz, senior manager of product innovation at Thomson Reuters, said "Our initial goal was to accommodate 2,000 events per second. Our tests show that Product Insight on AWS can process up to 4,000 events per second, and within a year we expect to increase that to more than 10,000 events per second." This figure represents more than 25 billion events per month. Even with this high throughput, the system has not lost any data since its inception. "Because of the robust failover architecture and the technical capabilities of AWS, we have not lost a single event since we started collecting data," says Fritz. For more information, see the Thomson Reuters Case Study[18] or watch the AWS re:Invent 2016 presentation Real-time Data Processing Using AWS Lambda (SVR301).[19]

# Big Data

AWS Lambda is a perfect match for many high-volume, parallel processing workloads. For an example of a reference architecture using MapReduce, see Reference architecture for running serverless MapReduce jobs.[20]

## Customer Example – Fannie Mae

Fannie Mae, a leading source of financing for mortgage lenders, uses AWS Lambda to run an "embarrassingly parallel" workload for its financial modeling. Fannie Mae uses Monte Carlo simulation processes to project future cash flows of mortgages that help it manage mortgage risk. The company found that its existing HPC grids were no longer meeting its growing business needs. Fannie Mae built their new platform on Lambda, and the system successfully scaled up to 15,000 concurrent function executions during testing. The new system ran one simulation on 20 million mortgages that completed in 2 hours, which is three times faster than the old system. Using a serverless architecture, Fannie Mae can now run large-scale Monte Carlo simulations cost effectively because it doesn't pay for idle compute resources. It can also speed up its computations by running multiple Lambda functions concurrently. Fannie Mae also experienced shorter than typical time-to-market because they were able to dispense with server management and monitoring, along with the ability to eliminate much of the complex code previously required to manage application scaling and reliability. For more information, see the Fannie Mae AWS Summit 2017 presentation SMC303: Real-time Data Processing Using AWS Lambda.[21]

# IT Automation

Serverless approaches eliminate the overhead of managing servers, making most infrastructure tasks, including provisioning, configuration, management, alarms/monitors, and timed cron jobs much easier to create and manage.

## Customer Example – Autodesk

Autodesk, which makes 3D design and engineering software, uses AWS Lambda to automate its AWS account creation and management processes across its engineering organization. Autodesk estimates that it realized cost savings of 98 percent (factoring in estimated savings in labor hours spent provisioning accounts). It can now provision accounts in just 10 minutes instead of the 10 hours it took to provision with the previous, infrastructure-based process. The

serverless solution enables Autodesk to automatically provision accounts, configure and enforce standards, and run audits with increased automation and fewer manual touchpoints. For more information, see the Autodesk AWS Summit 2017 presentation SMC301: The State of Serverless Computing.[22] Visit GitHub to see the Autodesk Tailor service.[23]

## Additional Use Cases

The use cases described in the previous section only scratch the surface of what's possible with Lambda and the other AWS serverless offerings. Other use cases include powerful human language understanding through chatbots built using Amazon Lex and AWS Lambda, low-latency global Edge Computing using Lambda@Edge with Amazon CloudFront, and powerful on-premises file processing with Lambda functions inside an AWS Snowball. These are just some of the exciting capabilities of this versatile approach. To learn more, see AWS Lambda.[24]

# Conclusion

Serverless approaches are designed to tackle two classic IT management problems: idle servers that drain a company's balance sheet without offering value, and the cost of building and operating fleets of servers, and server software, that distract and detract from the business of creating differentiated customer value. AWS Lambda and the other AWS serverless offerings solve these longstanding problems by eliminating the servers, containers, disks, and other infrastructure-level resources from the programming and billing model. As a result, developers can work with a clean application model that helps them deliver faster and organizations only pay for useful work. The easiest and fastest way to architect reactive, event-based systems and to deliver cloud-native microservices is through the use of serverless architectures. To learn more and read whitepapers on related topics, see Serverless Computing and Applications.[25]

# Contributors

The following individuals and organizations contributed to this document:

- Tim Wagner, General Manager of AWS Serverless Applications, Amazon Web Services

# Further Reading

For additional information, see the following:

- Serverless Reference Architectures with AWS Lambda by Werner Vogels, CTO at Amazon.com[26]
- AWS re:Invent 2016: The State of Serverless Computing (slide presentation) by Tim Wagner, General Manager of AWS Serverless Applications[27]
- The economics of serverless cloud computing by Owen Rogers, Research Director at 451 Research[28]

# Reference Architectures

- Web Applications[29]
- Mobile Backends [30]
- IoT Backends[31]
- File Processing[32]
- Stream Processing[33]
- Image Recognition Processing[34]
- MapReduce[35]

# Document Revisions

| Date | Description |
|---|---|
| **October 2017** | First publication |

# Notes

[1] https://www.forbes.com/sites/moorinsights/2016/04/11/tco-analysis-demonstrates-how-moving-to-the-cloud-can-save-your-company-money/

http://www.cloudstrategymag.com/articles/86033-understanding-tco-cloud-economics

[2] In 2012, Gartner estimated data center utilization ran from 7 to 12% (http://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html). A 2008 McKinsey study placed it at 6% (https://www.sallan.org/pdf-docs/McKinsey_Data_Center_Efficiency.pdf). An Accenture paper analyzing a set of EC2-based applications found approximately 7% utilization (http://ieeexplore.ieee.org/document/6118751/). A 2014 study from NRDC and Anthesis found that, in 2013, over 30% of servers were fully "comatose" (plugged in, but doing nothing of value) (http://anthesisgroup.com/wp-content/uploads/2015/06/Case-Study_DataSupports30PercentComatoseEstimate-FINAL_06032015.pdf).

[3] Occupy the Cloud: Eric Jonas et al., *Distributed Computing for the 99%*, https://arxiv.org/abs/1702.04024.

[4] https://aws.amazon.com/serverless/developer-tools

[5] https://aws.amazon.com/lambda/edge/

[6] https://aws.amazon.com/serverless/

[7] https://d0.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Architectures.pdf

[8] https://github.com/awslabs/lambda-refarch-webapp

[9] https://github.com/awslabs/lambda-refarch-mobilebackend

[10] https://aws.amazon.com/solutions/case-studies/bustle/

[11] https://github.com/awslabs/lambda-refarch-iotbackend

[12] https://www.slideshare.net/AmazonWebServices/aws-reinvent-2016-serverless-iot-back-ends-iot401

[13] https://www.youtube.com/watch?v=gKMaf5E-z7Q

[14] https://github.com/awslabs/lambda-refarch-streamprocessing

[15] https://github.com/awslabs/lambda-refarch-fileprocessing

[16] https://github.com/awslabs/lambda-refarch-imagerecognition

[17] https://www.youtube.com/watch?v=AcGv3qUrRC4&feature=youtu.be&t=1153

[18] https://aws.amazon.com/solutions/case-studies/thomson-reuters/

[19] https://www.youtube.com/watch?v=VFLKOy4GKXQ&feature=youtu.be&t=1449

[20] https://github.com/awslabs/lambda-refarch-mapreduce

[21] https://www.slideshare.net/AmazonWebServices/smc303-realtime-data-processing-using-aws-lambda/28

[22] https://www.slideshare.net/AmazonWebServices/smc301-the-state-of-serverless-computing-75290821/22

[23] https://github.com/alanwill/aws-tailor

[24] https://aws.amazon.com/lambda/

[25] https://aws.amazon.com/serverless/

[26] http://www.allthingsdistributed.com/2016/06/aws-lambda-serverless-reference-architectures.html

[27] https://www.youtube.com/watch?v=AcGv3qUrRC4

[28] https://451research.com/report-short?entityId=92764

[29] https://github.com/awslabs/lambda-refarch-webapp

[30] https://github.com/awslabs/lambda-refarch-mobilebackend

[31] https://github.com/awslabs/lambda-refarch-iotbackend

[32] https://github.com/awslabs/lambda-refarch-fileprocessing

[33] https://github.com/awslabs/lambda-refarch-streamprocessing

[34] https://github.com/awslabs/lambda-refarch-imagerecognition

[35] https://github.com/awslabs/lambda-refarch-mapreduce