< Back to News (https://www.globallogic.com/serverless-architecture-evolution-of-a-new-paradigm/)
Insights (https://www.globallogic.com/news-and-insights/) /
White Papers (https://www.globallogic.com/news-and-insights/papers)

Serverless Architecture: Evolution of a New Paradigm

Gyanendra Rai, Prashant Pasricha, Rakesh Malhotra, Santosh Pandey - May 24, 2017

Table of Contents

Introduction

Serverless Architecture – in literal words means an architecture without servers. Well, is that technically possible? Without servers, there cannot be a way to serve the requests coming from any client application. Even if we abstract the physical hardware behind virtualization and IaaS / SaaS vendors, someone still has to maintain servers somewhere.

Serverless Architecture may not be a precise technical term, but in essence, it is an industry movement towards fully modularized applications in the era of utility computing.

With Serverless Architecture, the code still technically runs on servers, but the user is not responsible for managing the servers. It is an essentially an amalgamation of self-managed functions and cloud services that serve as the Serverless application.

This whitepaper provides an overview of Serverless Architecture, its benefits and trade-offs, along with looking at what encompasses the Serverless architecture along with discussing a reference architecture.

Serverless Architecture

In today's competitive world, software is at the core of every business's ability to create products, services and offerings. The demand for innovative, technology-oriented, agile software solutions that are well integrated and self-sufficient is

ever increasing, and these modernizations are becoming key differentiating factors in a competitive landscape.

These innovative and modern solutions bring alongside with them some areas of concerns. One of the biggest considerations is the effort involved in the DevOps strategy with respect to management of servers, capacity planning, scalability, high availability etc. It is an important and critical area as it affects how fast our products/services can be rolled out to market and how they can scale with the volume growth.

Moving the software on cloud vendor provided infrastructure makes the infrastructure readily available, but one still needs to plan for capacity, disaster recovery etc.

Some other points going against the traditional deployment architectures is the maintenance and monitoring needs of the servers, continuous patching to keep them up to date against any unforeseen security threats, need of appropriately skilled resources to manage infrastructure, licensing costs of the software etc.

Serverless Architecture is the answer to addressing some of the above considerations. Serverless is a style of application architecture which allows focusing on the business logic without worrying about the environmental or infrastructure related concerns. It eliminates the need to manage servers on cloud, and replacing them with pervasive use of functional computing resources. In the next sections, we will explore on how Serverless Architecture handles these aspects.

Origin and Growth

The origin of Serverless goes back to the early 2000's when cloud computing vendors started providing open-source software and infrastructure setup to IT organizations, as hosted services on private or hybrid clouds.

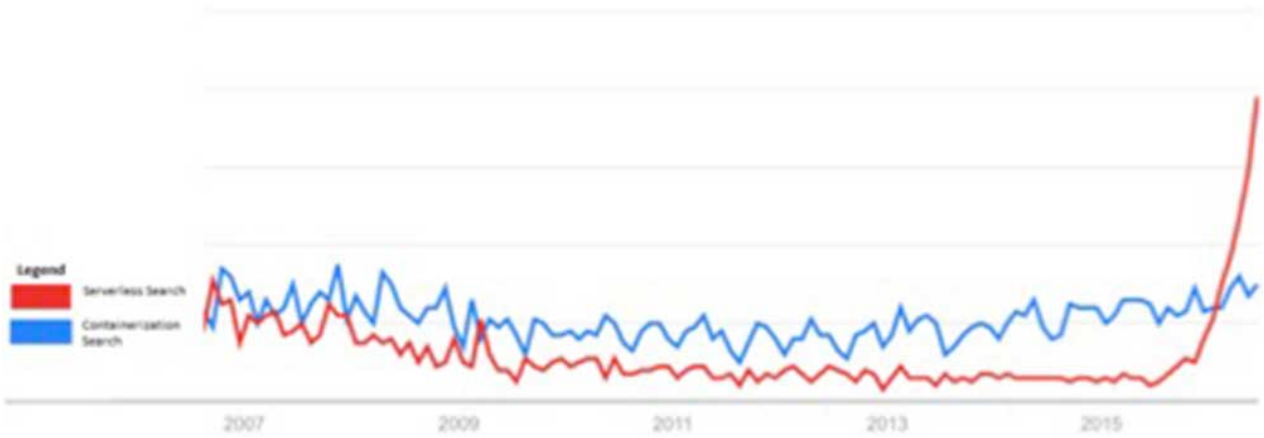
The framework matured after standardization of cloud-computing models, namely, Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Cloud providers also started providing hardware and software assets as per-use service-based models, resulting in significant reductions in costs when compared to existing expenditures of the IT organizations.

In 2014, Amazon Web Services released a groundbreaking service called AWS Lambda that offered a new way to deploy web, mobile or IoT application's code to the cloud. Instead of deploying the entire codebase of the application all at once, Lambda allows to deploy application's functions individually, to their own containers.

Microsoft announced its answer to AWS Lambda by launching "Azure Function" at its Build 2016 conference.

Google released "Cloud Function" as its own Serverless service in February 2016, but it is still in Alpha phase.

Serverless, originally a market limited to AWS's Lambda service, has expanded dramatically in interest because of competitive and upcoming services from Microsoft to Google to IBM.



The above graph shows the comparison of serverless search against containerization. The blue line represents the searches made for the containerization, whereas the red color line represents the web searches made for Serverless. *[1]*

FaaS – The Backbone of Serverless Architecture

Function as a Service (FaaS) is a type of cloud computing service that provides a platform that allows customers to develop, run, and manage application functionalities without the complexity of building and maintaining the infrastructure. Building an application with this model is the core of "Serverless" architecture.

FaaS is a recent development in cloud computing. AWS first implemented the concept and launched AWS Lambda in 2014, followed by Google Cloud Functions and Microsoft Azure Functions that are now available for public use.

FaaS functions do not require coding to a specific framework or library, and are regular applications when it comes to language and environment. All aspects – from execution to scaling are completely automatic, elastic, and managed by the provider.

Following are the key components of FaaS:

- Functions
- Events
- Resources

A **Function** is an independent unit of deployment, like a microservice. The responsibilities of a function may encompass:

- Saving a user to the database
- Processing a file
- Performing a scheduled task

Anything that triggers execution of a Function is regarded as an **Event**. It could be:

- A call to a registration service HTTP endpoint
- A file upload in the File server
- A message published in the Message Queue

Resource refers to the infrastructure or the components used by the Functions. For example:

- Database services (e.g. for saving Users/Posts/Comments data)
- File System services (e.g. for saving images or files)
- Message Queue services (e.g. for publishing messages)

Back to Table of Contents

Function as a Service (FaaS) Providers

This section focuses on the leading Function as a Service (FaaS) providers, the strengths displayed by the providers and the cautions that must be kept in mind before finalizing the provider for supporting a Serverless environment.

AWS Lambda Functions

A lambda function, in any language, is an anonymous function that accepts a single variable input, and is used for limited use to describe logic with a simpler syntax than a regular function.

On similar lines, AWS Lambda function is a simple function taking single input and performing required transformation and simpler/complex operations. They are configured to be fired in response to an event, such as files uploaded to an S3 bucket, HTTP requests using API Gateway etc. The functions must be written in a "stateless" style; to enable AWS Lambda to rapidly launch multiple copies of the function as needed to scale to the rate of incoming events. AWS Lambda functions can include libraries, even native ones.

AWS Lambda can be integrated with Amazon API Gateway, Amazon VPC, CloudFront, Identity Management, S3, RDS etc. that allows building a Serverless application that is highly available, scalable, and secure.

Originally designed for simpler use cases, like image uploading and tracking of website clicks, AWS Lambda can now support use-cases like auto-provisioning of back-end services linked to custom HTTP request events, and handling authentication and authorization by integration with AWS Cognito.

AWS Lambda runs the back-end code on its own fleet of Amazon EC2 instances across multiple availability zones, leading to high availability, performance and scalability of the AWS infrastructure.

There have been a few offered ways to run code in AWS cloud: Amazon EC2, Amazon ECS, and AWS Elastic Beanstalk. EC2 is usually a full-blown IaaS while ECS acts as a hosted container environment and Beanstalk is utilized as a PaaS offering. AWS Lambda is the newest in the lot – its innovative pricing mode based on request count, execution time, and allocated memory makes it an attractive option for moving web applications to cloud. *[2]*

Microsoft Azure Functions

Azure Functions provide a way to write a code block in cloud and execute it on demand. User needs to focus only on actual logic of code and its functionality rather than worrying about infrastructure to execute the code. It is an event based Serverless compute experience.

Azure Functions also provide integration with various services, including Azure and 3rd party services. These services can either trigger the function or serve as input and output for the Azure function. Some of the services are Azure Notification services, Service Bus, Document Services and so on.

Azure Functions support 2 different hosting plans:

1. App Service Plan

- Azure Functions run on a dedicated VM and may share the server with other apps running in the user's account
- Users pay for the hosted VM regardless of the utilization

1. Dynamic Service Plan

- Azure Functions run in parallel across app instances that automatically scale according to the utilization
- Users pay only for the instance execution time

Azure Functions are language agnostic and can be logically grouped together as per need to share the same resources and process, to provide a simpler way to manage and organize multiple functions in Azure.

Azure Functions are a great solution for processing data, integrating systems, working with internet-of-things (IoT), building simple APIs and microservices. Some other common business scenarios for Azure Functions include image processing, file maintenance and scheduled job processing.

Google Cloud Functions

Google Cloud Functions is an offering from Google Cloud platform to allow developers to create small, single-purpose functions that respond to cloud events without the need to manage a server or a runtime environment.

These functions are executed within a fully managed Node.js runtime environment that implicitly handles the underlying servers, operating systems and runtimes, so that the developer can fully focus on building the solution.

Some possible use-cases of Google Functions include:

- Convert a document uploaded to Google Cloud storage into a PDF
- Convert images uploaded to Google Cloud storage into thumbnails
- Scan and validate uploaded documents and send response immediately to the uploader

Google Functions can be scaled automatically, and can asynchronously process events emitted from other Cloud Platform resources, such as internal events by Google Cloud Storage or events through Google Cloud distributed message bus. For e.g. Google Cloud Function can be triggered when messages are sent to a topic by the publishers. HTTP calls are natively supported to enable synchronous, on-demand execution of functions. The two broad categories of Google Functions are as follows:

Foreground (HTTP) Functions:
HTTP functions invoked by HTTP(s) requests, which supports Request and Response mechanism of communication.

Background Functions:
Invoked when any event happens on Google Cloud storage or via a message on a Google cloud Pub/Sub offering.

Google Cloud Functions, though in alpha phase, looks very promising as they continue to work on a number of additional features to mark their strong presence in Serverless computing and to provide a direct competition to already established players like AWS and Azure functions.

Iron Functions

Iron.io is an open source, serverless application platform based on microservice architecture and docker containers, which provides rich features for job processing, build and deployment.

IronFunctions, based upon the Serverless platform provided by Iron.io, is a major release from Iron.io for processing of both synchronous and asynchronous functions. IronFunctions run on top of the orchestration frameworks like Kubernetes or Mesosphere, and are packaged using Docker such that they can support any language, any dependencies, and can run anywhere without being locked into a specific provider. IronFunctions are lambda compatible and have been written in GO, which is a powerful and high-performant language.

Initially, IronFunctions used to utilize a new container to handle every job, which led to an overhead per job due to container startup time. With its latest release, Hot Functions, IronFunctions now re-uses the permanent network connections such that long-lived containers serve the same task without incurring the startup time penalty. This update has vastly improved the throughput of IronFunctions (up to 8x depending on the duration of task).

IronFunctions implements a HTTP-like protocol to operate hot containers, but instead of communication through a TCP/IP port, it uses standard input/output. The basic cycle of executing these functions involves three steps: reading standard input, processing the work, and writing the output to stdout. Once done, the function is packaged by creating a Docker image for the function with an entry point, and pushing the Docker image to the Docker Hub registry for utilization.

FaaS Providers – Feature Comparison

The below table provides a single view comparison of the various features supported by the FaaS providers and how do they rate when compared to each other.

	AWS Lambda Function	Google Cloud Function	Iron Function	Microsoft Azure Function
Origin	2014	2016	2010	2016
Scalability & Availability	Automatic scaling	Automatic scaling	Automatic scaling	Automatic scaling
Max # of functions	Unlimited	20 functions per project (Alpha)	Within memory size of 320-2048 MB	Based on type of event triggers and available resources
Concurrent Executions	Defaults to 100 parallel executions per account per region	Unlimited	Within memory size of 320-2048 MB	One or more functions per App service
Supported Languages	JavaScript, Java and Python And Node.Js as runtime	JavaScript	All major languages – PHP, Python, Ruby, Node.js, Java, .Net, Go, Scala, binary executables	C#, F#, Node.js, Python, PHP, batch, bash, or any executable
Event-driven Architecture	Event Sources (S3, SNS, SES, Dynamo DB, Kinesis, Cloud Watch)	Cloud Pub/Sub or Cloud Storage Object Change Notifications	AWS-supported event as well as events from any services that supports webhooks	Azure and 3rd-party services
Deployment	Only ZIP upload (to Lambda or S3)	ZIP upload, Cloud Storage or Cloud Source Repositories	ZIP upload	Git Integration, KUDU REST API and MSdeploy.exe / WAWSDeploy.exe
Dependencies	Deployment Packages	npm package.json	Deployment Packages	NuGet and NPM
Organization	Generally Independent	Generally Independent		Grouped logically into an application
Memory allocation	Per function	Not specified		Per App service
Licensing	Closed Source	Open Source (Runtime)	Open Source (Runtime)	Open Source (Runtime)
Billing Models	Pay-as-code-executes	Pay-as-code-executes	Pay-as-code-executes	Pay-as-code-executes

Back to Table of Contents

Serverless Frameworks

The Serverless frameworks provide wrapper services and templates built over FaaS offerings to provide a vendor agnostic & simplified way of defining and setting up Serverless applications for the end user. For example:

- Pre-defined templates to define Serverless resources with only a few lines of text

- CLI to simplify the process of packaging & deploying a Serverless application

This section talks about some of the available serverless frameworks that help in defining & deploying Serverless applications.

Serverless.com

Serverless provides an open source framework to *"build auto-scaling, pay-per execution, event driven apps"* without compromising the core theme of Serverless computing. The framework currently only supports AWS Lambda with plans to support the other major cloud providers in future.

Key Characteristics

- Support for multiple languages (Python, Node.js, Java and more)
- Command Line Interface (CLI) to manage and build Serverless architecture in a simpler manner
- Support for pay when the code runs model
- Provide structure for ease in management of code, resources and events across large teams
- Documentation and Community Support

In Serverless.com framework, a service is a unit of organization for separation of concerns and to maintain all the functions in the project, the events that trigger them, as well as the resources that are utilized during the course of execution.

Serverless.com utilizes serverless.yml file for maintaining functions, events and the related resources. At the time of deployment, the Serverless framework translates all syntax in serverless.yml to a single AWS CloudFormation template to provide the users of the Serverless framework the safety and reliability of CloudFormation. *[3]*

APEX

Apex is used for managing Serverless applications intended for AWS Lambda. It provides a user-friendly command-line-interface (CLI) to create Serverless projects, add functions, and deploy on AWS. A JSON configuration file is used to define various aspects of a Serverless application. Apex also provides workflow related tooling support for testing functions, rolling back deployments, viewing metrics, tailing logs, hooking into the build system etc.

Key Characteristics

- Support for languages that are not natively supported by Lambda, e.g. Golang
- Environment variable population via command-line or initial configuration files
- Support for multiple environments via project.ENV.json and function.ENV.json files
- Command-line function invocation with JSON streams
- Command & function name auto completion
- Project bootstrapping with optional Terraform support
- Function rollback support
- Concurrency support for quick deployments
- Function metrics and cost analysis

AWS – SAM

AWS-SAM, formerly known as project Flourish, is a standard application model for Serverless applications hosted in AWS Lambda. It is an open source project used to define various parts of a lambda application e.g. Amazon API Gateway APIs, AWS Lambda functions, and Amazon DynamoDB tables.

AWS SAM utilizes CloudFormation template (http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/gettingstarted.templatebasics.html) to deploy the application as a CloudFormation stack (http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/updating.stacks.walkthrough.html), and defines a set of objects that can be included in a CloudFormation template to describe common components of Serverless applications easily.

The templates describe a Serverless application in accordance with AWS SAM JSON (http://www.json.org/) or YAML (http://yaml.org/spec/1.1/) formatted text files, and introduces several new resources and property types that can be embedded into the r (http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/resources-section-structure.html)esources section of the template. The templates may include all other template sections and use CloudFormation intrinsic functions (http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/intrinsic-function-reference.html) to access properties available only at runtime.

Key Characteristics

- Simplified process of packaging Serverless application using CLI

- Available under Apache 2.0 commercial friendly license for AWS partners to adopt and extend within their own toolset
- Support for API declaration and documentation using Swagger file
- Function rollback support
- Support for pay-as-you-use model

Back to Table of Contents

Serverless Benefits & Tradeoffs

This section discusses some of the benefits and tradeoffs of Serverless Architecture.

Benefits

No Servers to Manage

Serverless Architecture allows focusing only on the code that matters to application and not surrounding or other undifferentiating aspects like designing for scalability, high availability, O/S management etc.

Cost Efficiency & Decreased Time-to-Market

Reduced amount of investment required for inception and building a new application. Subscription packages provided by Serverless providers ensure that you use only those compute resources that are needed and do not pay anything when the code is not executing.

Step Ahead of PaaS

With PaaS, one still needs to think about scale. Even if PaaS is setup to allow application to auto-scale, it cannot be done at the level of individual requests. However, with Serverless framework, the provider takes care of finding a server where the code is to run, and to scale up when necessary. The containers used to run these functions are decommissioned as soon as the execution ends, and the user being charged only for the resources consumed when the code is run.

Reduced Installation & Deployment Complexity

Packaging and deploying a simple application developed for a Serverless framework involves a compile and upload operation, and is much simpler when compared to traditional packaging & deployment done for an entire server.

Improved User Experience, Geolocation, and Reduced Latency

Serverless has ability to become a standard approach to offloading functions to execute closer to end users and devices. With Serverless, providers have points of

presence near every user, and apps perform equally well for everyone.

Trade-Offs

Third-Party Vendor Locking

Another trade-off for utilizing Serverless Architecture is the dependence on third party vendors and services, leading to lack of control on concerns like system downtime, unexpected limits, cost changes, loss of functionality, forced API upgrades etc.

The Hidden Costs

While Serverless may lead to reduced operations costs, development costs and application complexity are increased. Projects that in the past would have been developed by small to medium sized teams, now require many small teams and significant product and team management complexity to move forward. *[5]*

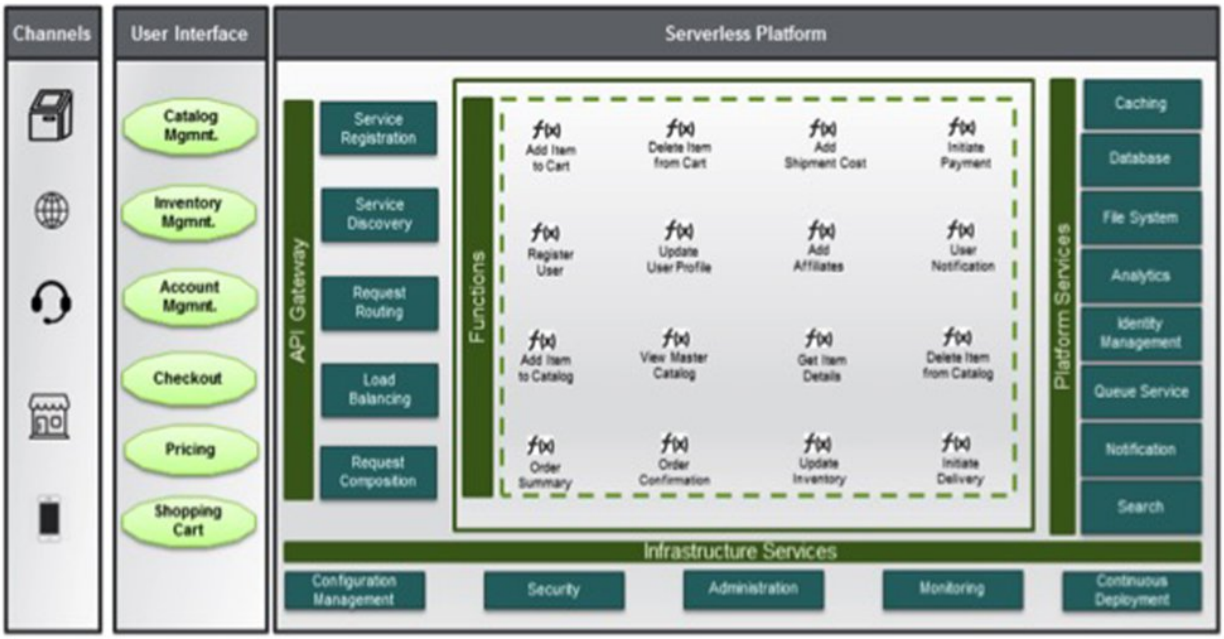
Increased Development and Testing Complexity

Serverless Architecture brings along with complexity of splitting a single application into a fabric of services and functions, which significantly increases with the number and variety of services. Similarly running integration tests and performing load testing for a Serverless platform is hard, since there is a dependency on externally provided systems for running testing scenarios.

Back to Table of Contents

Reference Architecture & Use Cases

This section depicts high-level overview of a reference e-commerce application based on Serverless Architecture. Also depicted below are some of the use cases where Serverless Architecture based on Function as a Service (FaaS) can be utilized to support real-time scenarios involving high volume transactions.



The following section gives a brief description on the layers, components and features of the depicted reference architecture.

Channels

The devices or communication media to interact with the e-commerce application., e.g.. Web application, mobile application etc.

User Interface

The presentation layer exposed to the end-users to perform relevant e-commerce operations, e.g. View catalog, view item details, add items to shopping cart, view prices, checkout etc. Each of the user actions will be a relevant event to trigger calls to the corresponding Function Services hosted with the Serverless providers.

API Gateway

Acts as middleware between the user requests and the serverless platform. This layer is the entry point for Function Services requests, simplifying access to the functions by routing the requests and the user data to appropriate functions. In addition, takes care of additional concerns like routing of requests between functions, security, transaction management, load balancing, etc.

Function Services

This layer is the core of the entire platform, and encapsulates the entire business logic for the unified solution in form of independent functions, hosted as services. These Function Services are triggered based on an event and perform the corresponding business logic associated with them. For example, a View

catalog event from the UI will trigger an associated function-as-a-service, which further fetches the catalog details from underlying database, and returns the results back to the user interface.

Platform Services

The underlying services provided by Serverless providers, which are utilized by functions to perform the expected business logic. For example, database services, caching services, file system services etc.

Infrastructure Services

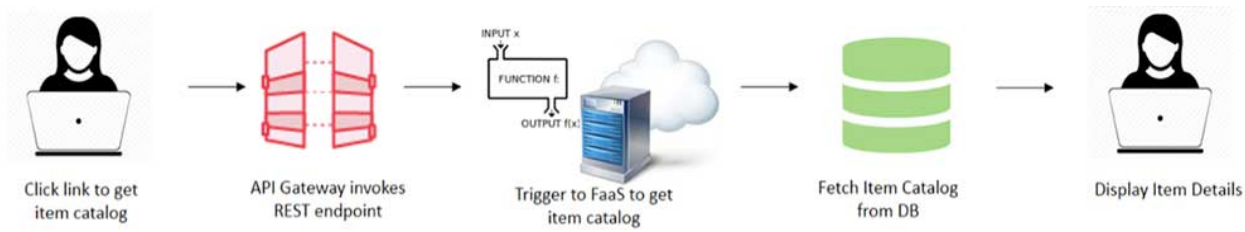
The additional services that are provided by the serverless platform to address the crosscutting concerns like monitoring, auditing, deployment etc.

Use Cases

Use Case 1. E-Commerce: Analytical Reporting of Online Orders

Use Case 2. E-Commerce: User Profile Updates

Use Case 3. E-Commerce: Fetch Latest Item Catalog



Back to Table of Contents

Conclusion

There is an ongoing shift and attention given towards Serverless Architecture. Utilizing offerings from Serverless providers, companies can now build large-scale platforms cost effectively. With Serverless Architecture, traditional servers are replaced with cloud functions that act as a discrete single-purpose service.

Having said the above, the introduction of Serverless Architecture by cloud providers has also set the stage for a level of technology and vendor lock-in that used to be a concern the early days of software development when Open Source was relatively unknown and organizations had to completely rely on vendors for their offerings. Therefore, the primary beneficiaries of this new architecture tend to be the cloud providers that are supporting Serverless Architecture.

In this current scenario, most organizations are conducting quick PoC's and implementing standalone functions as a service, utilizing the Serverless Architecture, . That said, the day when reference use-cases of Serverless architecture turn into actual running implementations, and start addressing the scales and volumes that everyone is looking out for, is not far down the road. The possibilities are endless.

Back to Table of Contents

References

[1] http://redmonk.com (http://redmonk.com)

[2] https://gigaom.com/2015/01/09/why-aws-lambda-is-a-masterstroke-from-amazon/ (https://gigaom.com/2015/01/09/why-aws-lambda-is-a-masterstroke-from-amazon/)

[3] https://serverless.com (https://serverless.com)

[4] https://github.com/awslabs/serverless-application-model/blob/master/versions/2016-10-31.md (https://github.com/awslabs/serverless-application-model/blob/master/versions/2016-10-31.md)

GlobalLogic®

(https://www.linkedin.com/company/globallogic) (https://twitter.com/globallogic) (https://www.facebook.com/GlobalLogic/) (https://www.youtube.com/user/globallogic) (https://www.globallogic.com/contact-us)