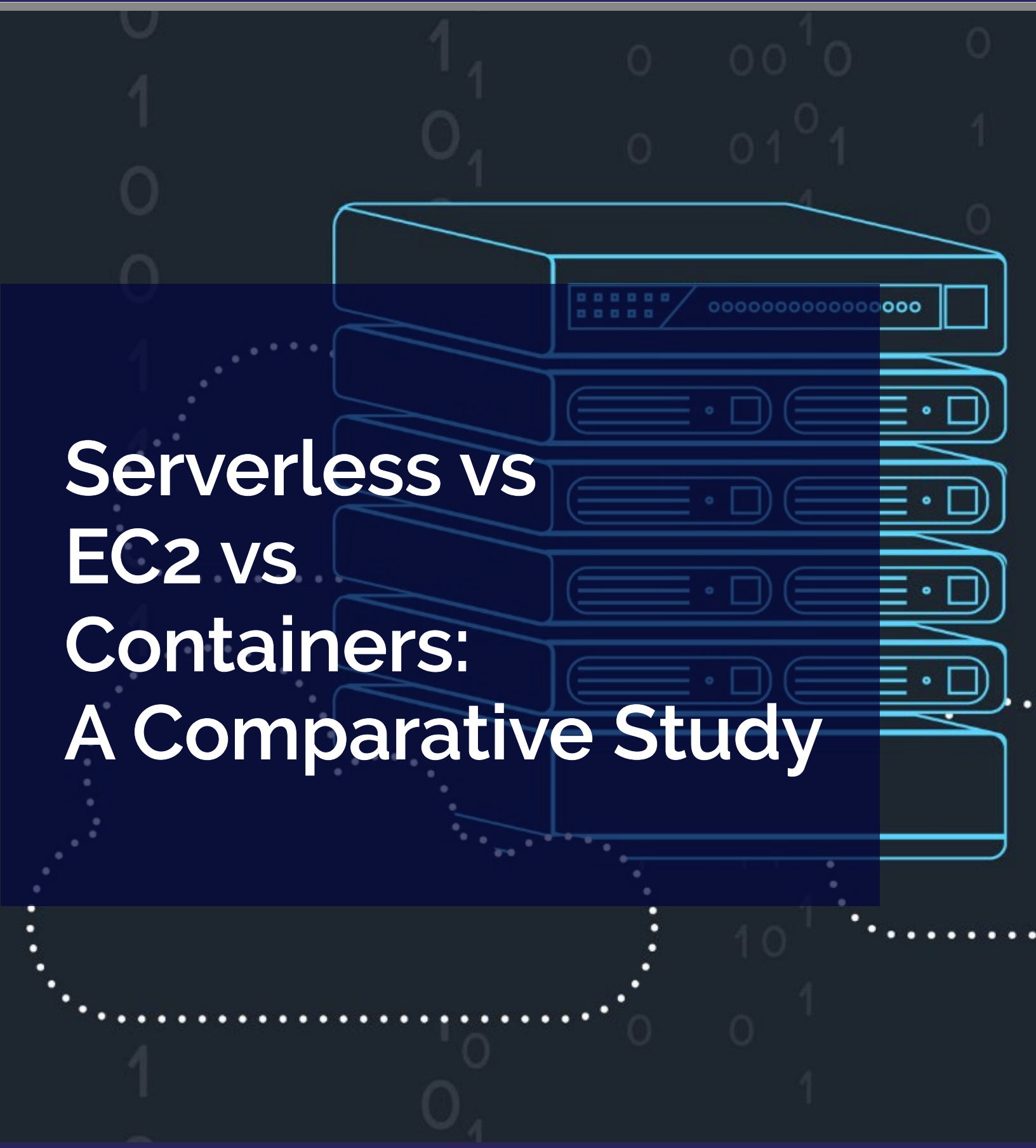
White Paper

Flux7
an NTT DATA Company

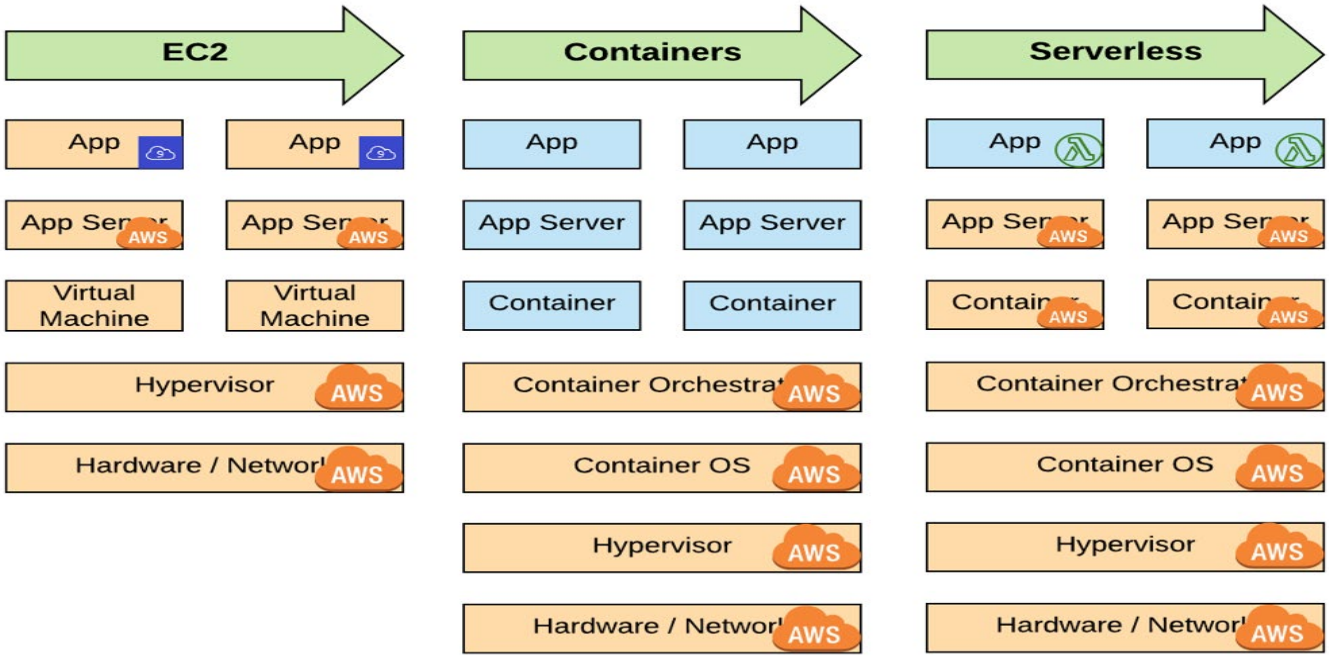
AWS Premier Consulting Partner - DevOps | Migration | Security



Serverless vs EC2 vs Containers: A Comparative Study

If you are looking for a Cloud Service framework for your business needs, then you have likely come across the terms Serverless, EC2, and Containers. However, they have unique differences that must be taken into consideration. Here, our container consulting team will compare these different models developers can choose from when deploying virtual infrastructure and applications. And, we'll discuss the benefits of each as well as some ideal serverless use cases.

As per Gartner, "serverless does not replace containers or VMs, but can support requirements for utility logic, unpredictable demand, and event-driven requirements." Further, Gartner predicts that this trend will become mainstream between 2020 and 2022, with 10% of IT organizations already using serverless computing.



Serverless: Serverless is an execution model where the cloud provider is responsible for executing a piece of code by dynamically allocating the resources. Serverless resources are priced by the amount of resources consumed; you pay only for what you use to run the code. And the granularity is finest with Serverless. Code typically runs inside stateless containers that can be triggered by a variety of events including HTTP requests, database events, queuing services, monitoring alerts, file uploads, scheduled events (Cron jobs), etc. Code that is sent to the cloud provider for execution is usually in the form of a function. Hence serverless is sometimes referred to as Function-as-a-Service or FaaS.

It is a common myth that Serverless means there are no servers as Serverless does not actually involve running code without servers. Rather, Serverless computing means that the business or person that owns the code does not have to purchase or rent servers for the code to run. Serverless computing eliminates infrastructure management tasks such as server or cluster provisioning, patching, operating system maintenance, security, and monitoring and capacity provisioning. Serverless continues to evolve since its launch in 2014, with services like AWS Lambda helping to continuously redefine the ways in which we develop and deploy software products.

EC2: Amazon Elastic Compute Cloud (EC2) is a virtual cloud infrastructure service offered by AWS. EC2 provides on-demand computing resources through which you can create powerful servers and applications in the cloud -- just as you would for on-premises infrastructure. With Amazon EC2, you have the ability to start and allocate virtual machines as needed for your application. It provides you with complete control of your computing resources and lets you run on Amazon's computing environments. Unlike Serverless, EC2 requires management and provisioning of the environment.

Containers: A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. Packaged with both the application and all the elements the application needs to run properly--including system libraries, settings, and all other dependencies--a container only needs to be hosted and run in order to perform its function.

In a container-based architecture, how to increase container scale is determined by the developer in advance. Containers can easily be moved around and deployed wherever required. Each container shares the machine's kernel with other containers but runs as if it is the only system on the machine.

Comparing Serverless, EC2 and Containers

Given this background, let's take a look at how the three frameworks compare in some basic areas important to developers:

	Serverless	EC2	Containers
Overview	Simple deployment is what sets serverless apart. With no administration of infrastructure, you simply upload your functions and get started. No Docker files or Kubernetes configurations are required. All of which result in a speedy time-to-market.	EC2 requires management and provisioning of the environment.	Container technology enables you to scale your applications as much as you want. With serverless, that's not always the case as serverless may cause size and memory restrictions.
Cost	Serverless is cheaper than containers. When an application is not being used, it shuts down, and you don't pay for the idle time. The granularity of pricing is most cost-effective in serverless.	Pricing of EC2 instances depends on the application's needs. For a high-compute use case, EC2 will be a good fit and cheaper generically when compared to serverless and containers.	Lower than EC2, generically.
Security	Serverless architecture removes the patching and manual OS update work. You have higher flexibility of workflows but at the same time, it increases the surface attack. With an increasing number of functions, monitoring becomes hard which in turn is a threat to decaying functions. Functions are automatically scalable which gives good protection against DDOS attacks.	With EC2, you'll have to take care of the security layer at the instance level. The security layer decides and controls the traffic allowed to communicate with each instance. Each instance can have multiple security layers that dictate the allowed inbound traffic through certain protocols like TCP, UDP, ICMP, etc.	You have full flexibility and control with containers in terms of setting policies, managing resources, and security.
Maintenance	Serverless applications can be live as soon as code is uploaded.		Containers take longer to set up initially than serverless functions because it is required to configure system settings, libraries, and so on.

Importance of Serverless

Serverless is clearly gaining traction within the industry in part due to its many benefits:

- Reduced cost/pay as you use model - If an application does not get a lot of traffic (common for non-production environments) then serverless is a cost-efficient model.

- Serverless architectures are easily scalable - Applications built with a serverless infrastructure will scale automatically as the user base grows or usage increases. Developers can choose a serverless architecture to release and iterate new applications quickly, without having to worry about whether or not the application can scale.
-
- Supports event triggers - If you just want to write some quick bits of code that can respond to events without going through the process of creating an entire app/API, then serverless is likely the best approach.

- Deployment time - Deployment time is exceedingly fast, measured in milliseconds.

- Automated high availability - Serverless provides built-in availability and fault tolerance. You don't need to architect these capabilities since the services running the application provide them by default.

- Decreased latency - Code can run from anywhere as the application is not hosted on an origin server. It is therefore possible, depending on the vendor used, to run application functions on servers close to the end-user, thereby reducing latency as user requests no longer have to travel to an origin server.

- Reduced administration - A serverless system won't require continuous integration/delivery or containerization tools. A fully serverless solution requires zero system administration.

- Lower risk of failure - Management, resolution, and operation are outsourced to a third-party vendor. This lowers risk as the third-party vendor is responsible to solve any issues.

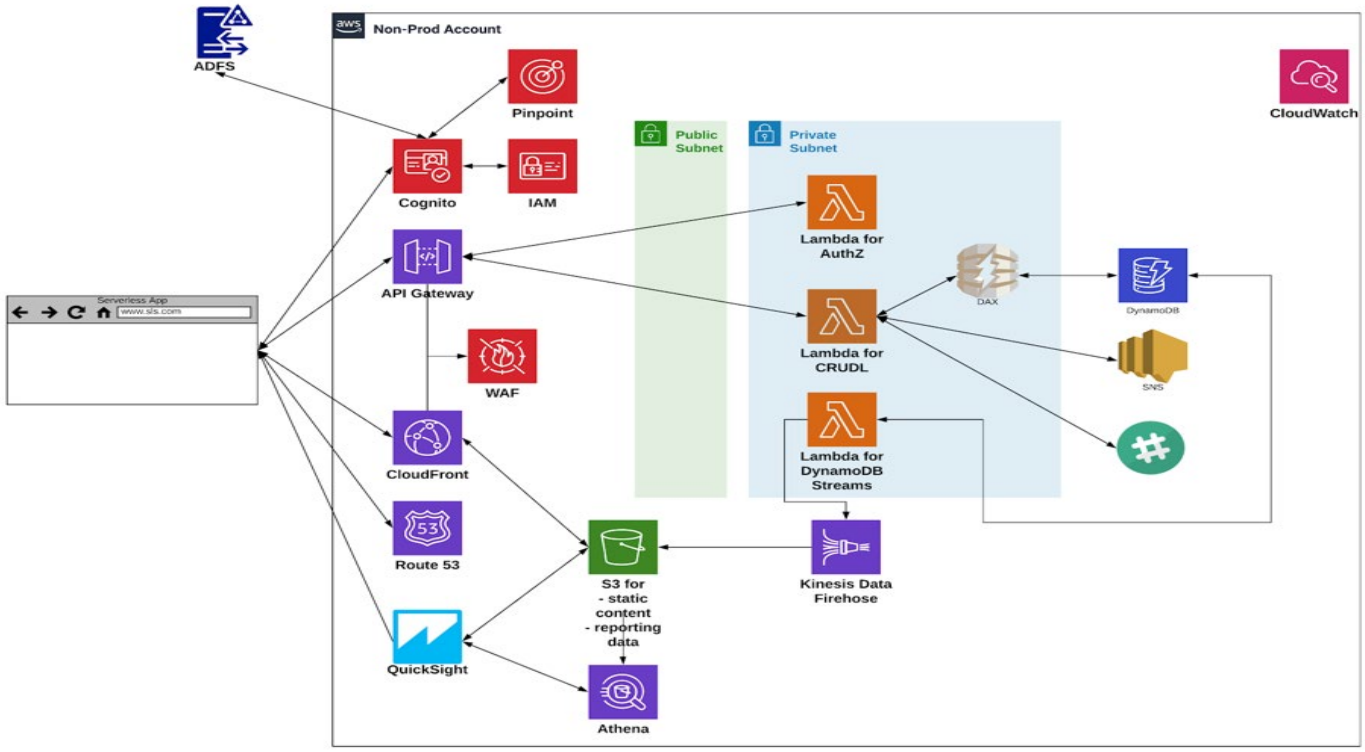
Use Cases

With that bit of background and an understanding of what sets serverless apart, let's look at a few generic, yet ideal serverless use cases to give you an idea of how serverless can address a wide variety of application problems.

Serverless Websites/Applications/Microservices

While basic websites are built using services like AWS API Gateway, DynamoDB, Amazon S3, and Amazon SQS, as can be seen in the graphic below, AWS Lambda, along with other AWS services, are used to build powerful serverless websites, applications, and microservices.

Continuous Integration and Continuous Deployment (CI/CD) Pipelines:



CI/CD pipelines allow DevOps engineers to ship code in small increments and serverless automates these processes. Code changes trigger website builds and automatic redeploys, or pull requests trigger the run of automated tests to make sure the code is well-tested before human review. Further, the build pipeline publishes source code into SonarQube, which in turn performs a static analysis of the code to detect bugs, code smells, and security vulnerabilities.

Event Streaming, an AWS Lambda Example

Serverless compute is triggered from sub-topics or from event logs, providing elastic and scalable event pipelines without the maintenance of complicated clusters. A simple use case of an AWS serverless application that receives a stream of events for two different clients changing states is given below. When events are pushed into Amazon Kinesis, Lambda triggers appropriate rules out of DynamoDB for each client and evaluates whether the rule is satisfied. If so, an alert is published to Kinesis Alerts Stream and Lambda triggers notifications.

AWS Lambda Serverless:

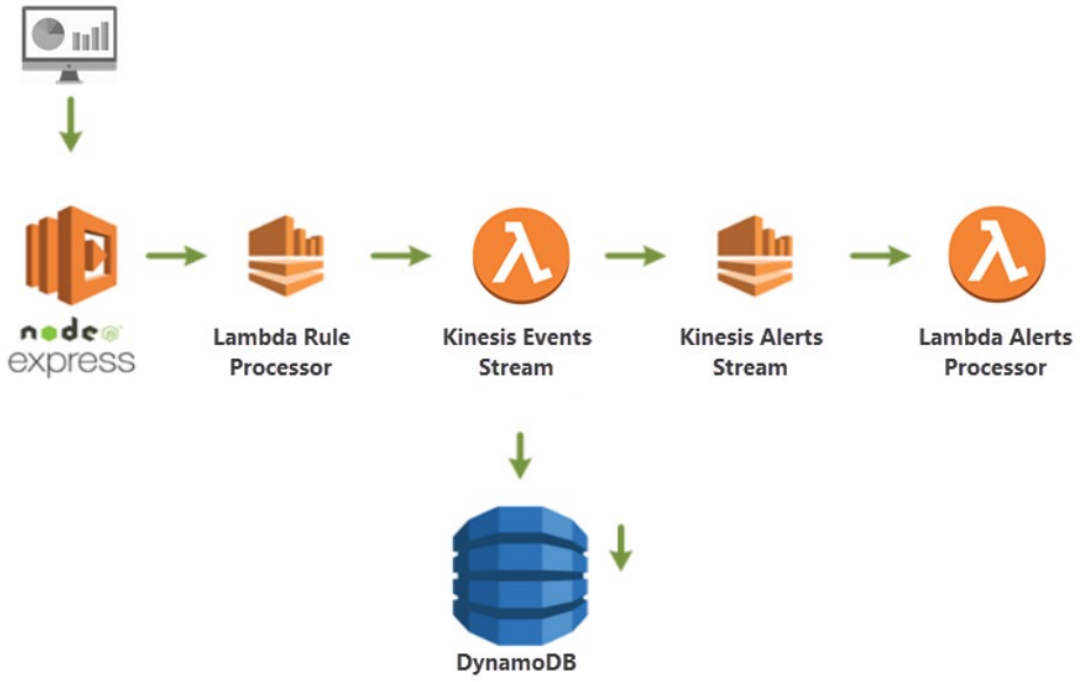


Image and Video Manipulation

A serverless video transcoding use case involves a Lambda trigger submitting a transcoding job using Amazon Elemental MediaConvert when a new file is ingested. The transcoding job specifies video packaging settings like HLS and DASH along with the bitrates that are expected from the transcoded output stored in another Amazon S3 bucket. Then, the videos are delivered securely by Amazon CloudFront, restricting access to the Amazon S3 bucket by using origin access identity (OAI).

Flux7 Implements Serverless for Automotive Leader

For a global auto manufacturer, Flux7 implemented AWS Lambda and AWS Step Functions to create accounts automatically; AWS Lambda and Step Functions are used to automatically create new AWS accounts under various organization units.

- The processing backend of the account creation system, Lambda functions are integrated with the API gateway and AWS Step Functions to create new accounts and organizational units.

- AWS Step Functions is a service that is used to coordinate the different parts of the account creation process; we can configure the Step Functions State machine using Amazon State machine language. Each task in the State machine is a Lambda function which is invoked in sequential order as defined using the state definition. Step Functions control the execution of the Lambda function, handling the input/output of each function and any errors raised from each task.

- The process begins with a request placed from the UI based application which triggers an API which in turn triggers a Step Function which then triggers a series of Lambda functions.

Ready to get started? Read our Tech Tutorial, "How to Start the Serverless Journey: Serverless Framework Deep Dive"

Serverless Decision Factors

There is no doubt that the serverless approach offers appealing benefits to software teams and customers, but it's not necessarily a fit for every deployment scenario. So before making decisions on serverless deployments, you should evaluate the following:

- Cloud architecture, performance, and reliability as the application should continue to function even when an availability region goes down.
- Vendor lock-in issues and dependencies as migrating and integrating functions and services pose new challenges.
- Tools and support as introspection and debugging are yet to mature.
- Distributed computing architectures as they are complex and consume time to assess, implement, test, and refactor.
- Memory and functional limitations as with serverless vendors pose limitations on the use of RAM and impose timeouts on functions.
- The skillset of your internal team and their ability to take-on serverless.

While the approaches discussed in this white paper tend to weigh toward serverless as the best option, it really depends on many factors such as the complexity of the application, pricing, choice of database, application adoption, hosting, and more. From a business perspective, the real attraction of serverless is giving developers space to quickly and easily experiment with new features and services, reducing time-to-market in an increasingly competitive world.

About Flux7

Flux7, an NTT DATA Company, is an IT services firm that helps enterprises reduce the complexities of a new or evolving cloud automation strategy. Agile and DevOps-native, Flux7's robust services portfolio prioritizes a fast path to ROI that meets the immediate needs of technical and innovation teams focused on transformation while forging a secure and stable pathway for security and operational excellence. Learn how Flux7 helps businesses bring solutions to market faster at https://www.flux7.com

Conclusion

As we've seen, there are many challenges when it comes to building successful IoT implementations — from the infrastructure through instilling good app design processes. And as the customer IoT success stories above have shown, an infrastructure that is not only secure but elastic and agile enough to deliver innovation must be in place.

About Flux7

Flux7, an NTT DATA Company, is an IT services firm that helps enterprises reduce the complexities of a new or evolving cloud automation strategy. Agile and DevOps-native, Flux7's robust services portfolio prioritizes a fast path to ROI that meets the immediate needs of technical and innovation teams focused on transformation while forging a secure and stable pathway for security and operational excellence. Learn how Flux7 helps businesses bring solutions to market faster at https://www.flux7.com