


Why Serverless will enable the Edge Computing Revolution

Serverless changes how we build applications both technically and conceptually — the same changes will enable Edge Computing adoption.



Ben Ellerby · Follow
Published in *Serverless Transformation* · 9 min read · Apr 19, 2021

👍 168

💬 1

🔖

🕒

📄

⋮



On the Edge

Edge Computing is a model in which computing and storage move closer to the end user. Currently ~10% of enterprise data is created and processed outside the traditional Data Centre/Cloud. **By 2025 that's predicted to reach 75%.**

Content Distribution Networks (or CDNs) represent the first wave of Edge computing. With a CDN the data needed by users is stored in multiple edge locations closer to the users — reducing the transport time and improving performance. As CDNs have become more advanced the maturity of virtualisation technologies has allowed code as well as storage to move outside the bounds of the Cloud/Data Centre. This has enabled computing resources to emerge as a service in Edge locations.

The typical usage of Edge computing is for real-time and instant data processing. As opposed to the Cloud, where “Big Data” is the name of the game, there is much more of a focus on realtime “Instant Data” as we move to the Edge.

At its core Edge computing is all about **reduction** — reduction in latency, traffic, bandwidth, geographic distance, energy and power. The reduction in latency brought by Edge computing can make certain applications in IOT, AI and ML more achievable. For instance: realtime instruction of autonomous devices, remote surgery and facial recognition are just a few emerging technologies that will leverage Edge computing to some extent.

The rise in Edge computing is further spurred on by the evolution of 5G and faster connectivity and devices in general. While Edge computing is reducing latency by bringing compute closer to user, 5G is reducing the latency of the communication. Together 5G and Edge computing can bring latency down to previously unattainable levels — allowing a new generation of application use cases.

What is Serverless?

Serverless enables us to build applications and services without thinking about the underlying servers. It's an imperfect name for a broad spectrum of services — but at it's core it's an architectural movement to increase agility through reduced Total Cost of Ownership (TCO).

In short, Serverless is an abstraction. We've abstracted away the details of the underlying machine to enable a model in which pure application code is sent to the Cloud Provider (e.g. AWS) and run in response to different events.

Lambda, the “poster child” for Serverless, is the most widely known Serverless Service — the Function as a Service (FaaS) from AWS. Lambda has changed how we can build application and the range of triggers for Lambda functions has enabled elegant event-driven architectures to flourish.

Technology — it's always cyclical

Take web applications as an example. Initially a lot of the logic lived on the server-side. When a web page was loaded a framework and templating engine would handle populating the dynamic parts of the page with the relevant data from a database. The resulting rendered HTML page would then be sent and displayed on the client side. The move to more and more complex user experiences saw the emergence of SPA frameworks like React and Vue, with a lot more business logic moving to the client side. As time went by we started to see the negatives of this — the user experience was slow and SEO was impacted. Frameworks like React “reinvented” SSR (Server-Side Rendering) — and we saw a move back to the rendering happening on the Server side.

The way we build entire applications has gone through a similar cycle. Initial computing happened on mainframes, then it moved to the client side with the rise of more powerful PCs, and now we see a move back to central cloud providers. We're about to undergo another cycle.... **a move to the Edge.**

Real World Edge Computing

Smart CCTV cameras, IoT devices, medical devices and CDNs (as discussed above) are all use cases of Edge computing. We're moving processing closer to the place the data is generated. In the case of CCTV this could be the use of instant fall detection through a model trained to recognise a person falling over — there can be almost zero latency in the inference of a fall event and in a factory setting this could halt machinery instantly.

Edge computing is not without its risks. It's hard to go a day without hearing about the security issues of “smart” devices with the rapid adoption of IoT across all sectors. The scalability of Edge computing services can mitigate some security threats like DDoS, but the core computing model has to adapt to operating in less trusted environments.

Worlds Collide — The Serverless Edge... CentreLess?

As stated above, Serverless is all about abstraction. Developers can write application code and ignore the details of the infrastructure. This abstraction, the pay-per-use pricing model and ephemeral (short-lived) execution model of FaaS solutions have a complementary conceptual model to the Edge world.

In the Edge world:

- We don't control the hardware — so abstraction is key
- It's an untrusted environment — so we need rigid sandboxing
- We're often processing unpredictable real-time data at scale — so we need an extremely scalable and elastic model for compute.

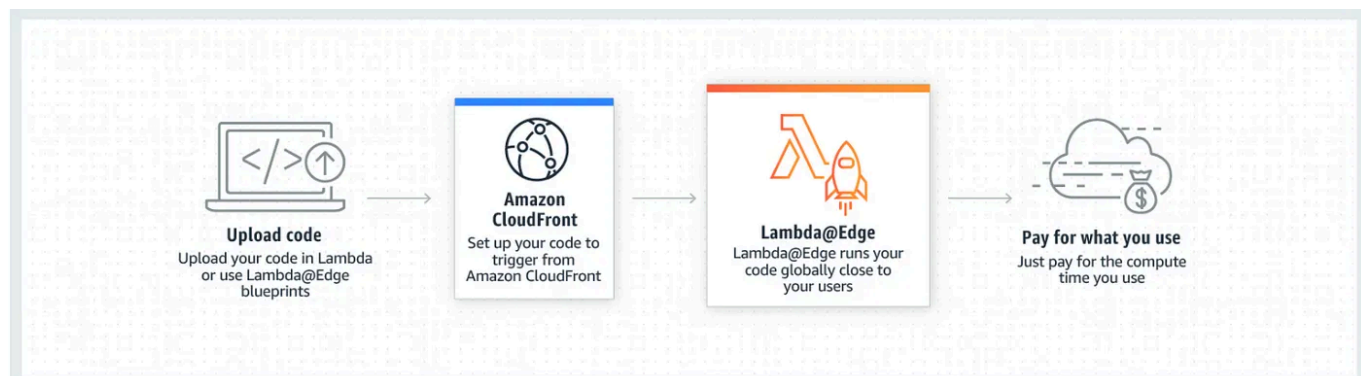
- The typical computers used at Edge locations need to be small and cheap, therefore we're operating in a low resource (compared to modern Server sizes) environment when it comes to CPU, RAM and disk — so we need to have compute efficiency and small memory/storage footprints.

All these points are challenges that were faced by Serverless, meaning the technology, conceptual model and best practices are complementary to Edge.

Serverless has matured extensively since its inception. The tooling, both from the vendor side and the open-source space, has improved massively. Along with this, practitioners have developed best practices and mental models for working in a Serverless environment. All of these advances will enable the rise of Edge computing.

Developers have got used to coding in a stateless and abstract runtime environment — as well as integrating with Serverless databases and third parties with elegant design patterns. The paradigm shift from a Serverless compute model to an Edge compute model is marginal, whereas going from a traditional Server or even Kubernetes environment is a much greater leap — both in mindset & tooling.

Lambda@Edge — Serverless services are already at the Edge



Lambda@Edge doesn't actually sit in the "Serverless" team of AWS. In fact it's a CloudFront (CDN) product. Lambda@Edge lets customers run application code closer to users by running in the CDN layer. It's very similar to "traditional" Lambda — we don't have to manage the infrastructure and we only pay when the code is running.

I often make use of Lambda@Edge for security checks, location routing and context specific data modification, SSR of React applications closer to the users, the classic image transformation use-case and even for some basic A/B testing.

With Lambda@Edge code must be deployed initially in the US-East-1 region — it's then distributed across 100s of data centres across the world. Surely distributing code to 100s of locations and running it in the CDN layer in response to network requests is going to be complicated?... well no, it's **pretty much the same as deploying a Lambda function** — which, thanks to years of Serverless community development, has become a simple task.

```
yaml
service: cloudfront-service

provider:
  name: aws
  runtime: nodejs10.x

functions:
  cflambda:
    handler: functions/handler.cloudfront
    events:
      - cloudFront:
          eventType: origin-request
          origin: https://app.acme.com
```

Serverless.yml file definition for a single Lambda@Edge function — the complete IaC template needed.

Using the Serverless Framework we can define our function, state it's triggered by a CloudFront event and then run a single `serverless deploy` command.

The code itself can be written in Python or JavaScript and there is no manual bootstrapping or virtualisation concerns to deal with. For instance, we could gradually migrate traffic from one S3 bucket to another in a few lines of code.

```
'use strict';

function getRandomInt(min, max) {
  /* Random number is inclusive of min and max */
  return Math.floor(Math.random() * (max - min + 1)) + min;
}

exports.handler = (event, context, callback) => {
  const request = event.Records[0].cf.request;
  const BLUE_TRAFFIC_PERCENTAGE = 88;

  /*
   * This Lambda function demonstrates how to gradually transfer traffic from
   * one S3 bucket to another in a controlled way.
   * We define a variable BLUE_TRAFFIC_PERCENTAGE which can take values from
   * 1 to 100. If the generated randomNumber less than or equal to BLUE_TRAFFIC_PERCENTAGE, traffic
   * is re-directed to blue-bucket. If not, the default bucket that we've configured
   * is used.
   */
  const randomNumber = getRandomInt(1, 100);

  if (randomNumber <= BLUE_TRAFFIC_PERCENTAGE) {
    const domainName = 'blue-bucket.s3.amazonaws.com';
    request.origin.s3.domainName = domainName;
    request.headers['host'] = [{ key: 'host', value: domainName}];
  }
  callback(null, request);
};
```

The "handler" function and it's parameters are very familiar to users of "traditional" Lambda Functions

The developer experience is the same as developing Serverless applications with Lambda. The only difference is that it needs to be deployed to US-East-1 first and there are more constraints on the execution time, CPU and memory. Also only JS or Python (no other supported languages or options for Custom Runtimes) can be used and deploying/rollbacks take longer due to distribution time.

This is a great example of how the execution model of Serverless is complementary to the execution environment of the edge.

Firecracker — solving many challenges for Serverless that Edge computing will face

AWS developed (and open-sourced) Firecracker as an enabling technology to Lambda & Fargate by improving speed and overall efficiency. Firecracker is a virtualisation technology that enables workloads to execute in "microVMs". microVMs are lightweight VMs (Virtual Machines) that provide security and isolation while maintaining speed and flexibility.

Security and low startup-times are enabled via a very minimal design. This ensures the memory footprint is kept low, they run quick and have a minimal attack surface.

In short, Firecracker brings greater tenancy density (number of microVMs per machine), improved security, reduced memory requirements and rapid startup times. This is achieved by the leveraging of Linux KVM (virtualization at the kernel level — with the kernel acting as a hypervisor), using Rust as a highly performant implementation language and a very specific and minimal API.

Firecracker aimed to improve latency in a resource constrained environment for the purpose of a Serverless FaaS (Lambda). The needs for low latency, strong security isolation and tenant density in a low resource environment are a massive overlap with the needs of Edge computing. Code will need to be run with low latency (especially due to the use cases highlighted above) in a low trust multi-tenant environment on small resourced machines.

It's not surprising then that Lambda@Edge runs on Firecracker. It's just one example of a technological advance made for Serverless that benefits the development of Edge computing.

Emerging Edge Computing Services

In addition to Lambda@Edge there are a number of other FaaS approaches to Edge Computing. Fastly & Cloudflare Workers are two great examples.

With Cloudflare Workers you can get a Serverless execution environment on a global network of V8 isolates. The code is run on Cloudflare's network (which is milliseconds from nearly all internet users at this point) and cold starts are sub 1ms. There's support for a range of runtimes (JS, Rust, C & C++) and access to Edge storage via the Cloudflare's Edge key-value store.

Storage is an interesting challenge for Edge based applications. Cloudflare have recently worked with Serverless Database providers like Fauna (a API driven transactional database that's completely Serverless) to improve Edge storage. In this way, Compute at Edge can be combined with “edge-first” databases. Edge applications will generally need both compute and state — and API based Serverless databases are filling the state side of that equation due to the distribution requirements.

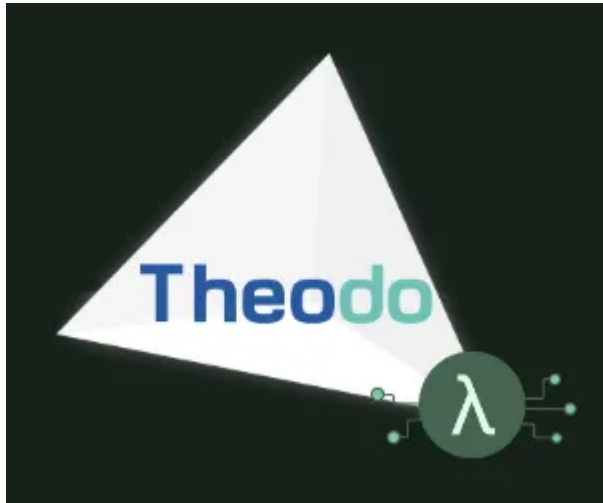
Fastly is another vendor providing Edge computing via its Compute@Edge service. As a CDN, Fastly is designed with Edge compute in mind, providing the ability to run code close to users as well as the general functions of a CDN. In a similar way to Cloudflare Workers, Fastly's Compute@Edge allows compute to operate at the CDN layer — bringing features like image optimisation, load balancing, security enhancements and geo-routing to existing and new applications.

Serverless will be the enabler for Edge

We've seen that Edge computing services are already making use of technologies developed for Serverless. We'll keep seeing this trend but it's nothing compared to the conceptual development Serverless has triggered.

Developers thinking in a stateless, pay-per-use, event-driven and distributed mindset is a paradigm shift that Serverless triggered. This paradigm shift will also be key to enable the adoption of Edge compute. Further to this, many of the technologies and best-practices we've been developing for Serverless will form the basis of those for Edge Compute.

Serverless changes how we build applications — these are the same changes the Edge Computing Revolution will need.



Serverless Transformation is a [Theodo](#) initiative. **Theodo** helps companies leverage Serverless through expert **delivery** and **training** teams. [Learn more about Theodo and speak to one of our teams today!](#)

If you like content like this consider subscribing to our [weekly newsletter!](#)

Serverless Edge Computing AWS Cloudflare Workers Fastly



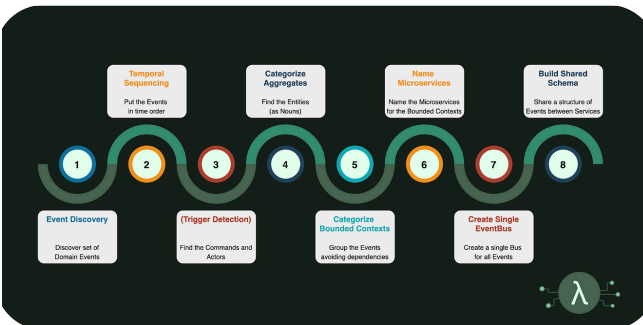
Written by Ben Ellerby

662 Followers · Editor for Serverless Transformation

Founder of aleios (<https://www.aleios.com/>) and AWS Serverless Hero

Follow

More from Ben Ellerby and Serverless Transformation



Ben Ellerby in Serverless Transformation

EventBridge Storming — How to build state-of-the-art Event-Drive...

Serverless Architectures should be Event-Driven and use Amazon EventBridge. Use...

10 min read · Apr 7, 2020

616 5



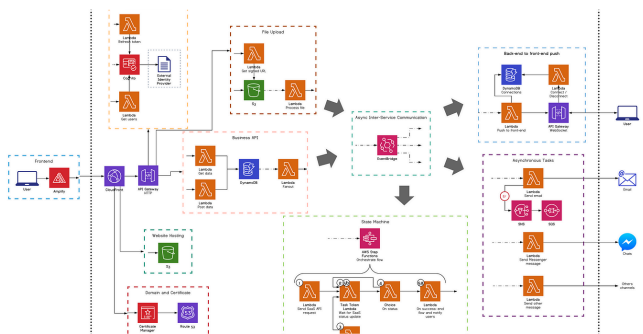
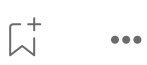
Xavier Lefevre in Serverless Transformation

Asynchronous client interaction in AWS Serverless: Polling,...

Event-driven Serverless often requires async update to the client. There are several ways ...

6 min read · Apr 18, 2020

730 7



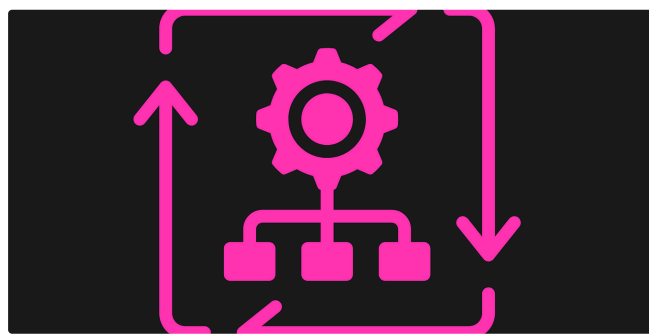
Xavier Lefevre in Serverless Transformation

What a typical 100% Serverless Architecture looks like in AWS!

If you are new to serverless and looking for a high level web architecture guide, you've...

11 min read · May 19, 2020

2.8K 19



Ben Ellerby in Serverless Transformation

Enabling the Optimal Serverless Platform Team — CDK and Team...

We want our Cloud teams to be secure, fast, stable, reactive, autonomous and of course.....

7 min read · Aug 10, 2022

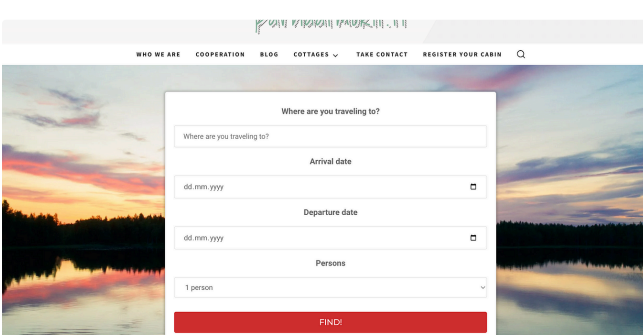
28 1



See all from Ben Ellerby

See all from Serverless Transformation

Recommended from Medium



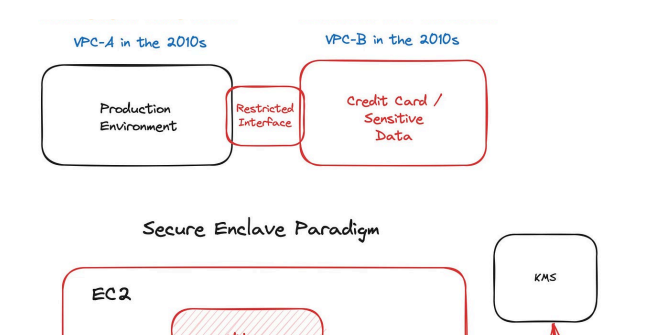
Artturi Jalli

I Built an App in 6 Hours that Makes \$1,500/Mo

Copy my strategy!

3 min read · Jan 23, 2024

12.4K 147



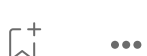
Dheeban SG

Exploring AWS Nitro Enclaves for Practical Web3 Use-Cases

A quick and simple guide to grasp this technology

3 min read · Oct 30, 2023

1 1



Lists