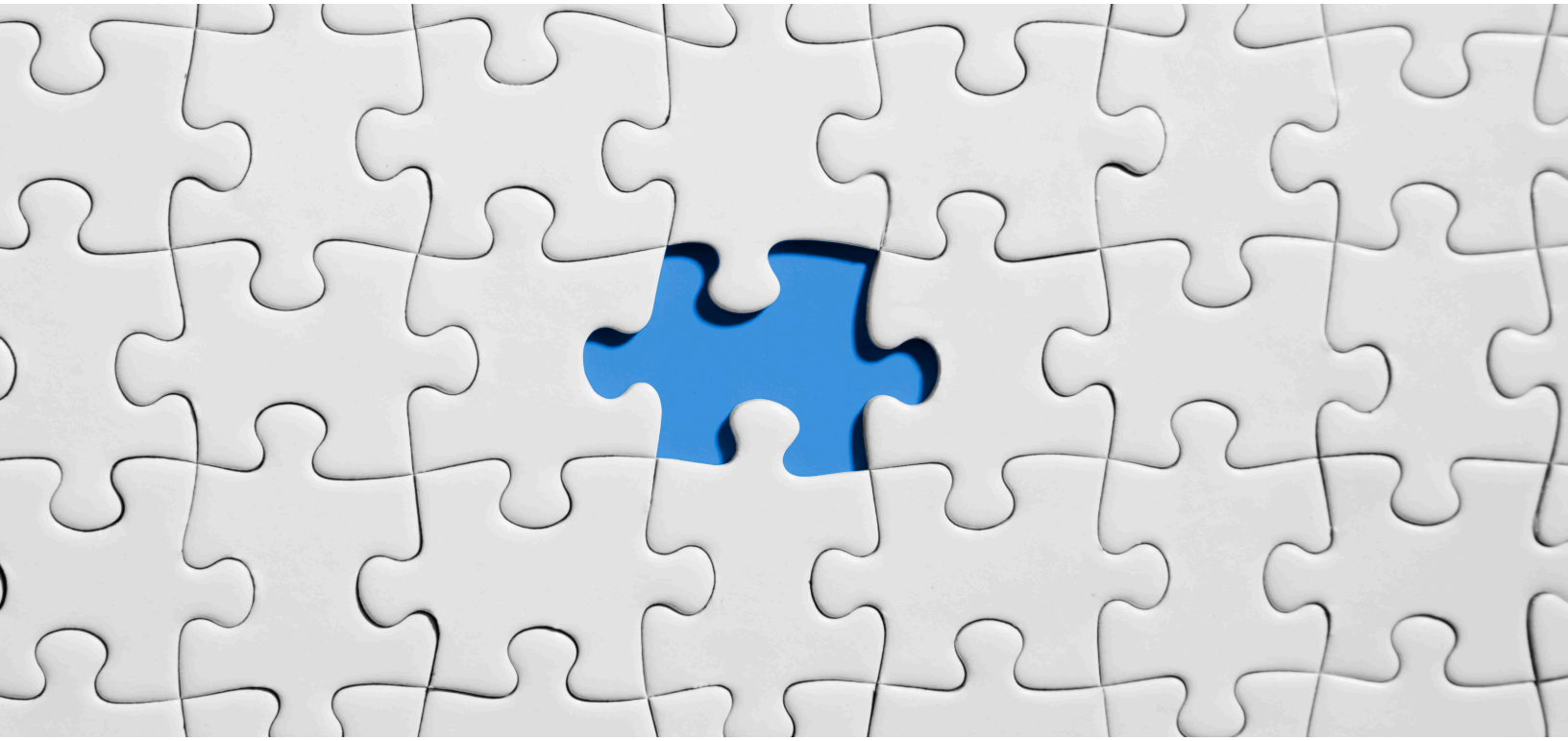
DELL Technologies

URBAN EDGE COMPUTING: SERVERLESS, BETTER, CHEAPER



Shadaab Mohiuddin

Senior Principal Systems Development Engineer

Boomi

Shadaab.mohiuddin@boomi.com

Tabraiz Amman

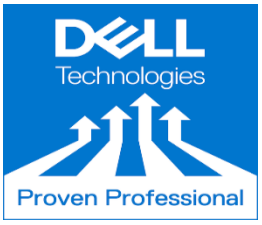
Senior Principal Systems Development Engineer

Boomi

Tabraiz.amman@boomi.com

Knowledge Sharing Article

DELL Technologies Proven Professional



The Dell Technologies Proven Professional Certification program validates a wide range of skills and competencies across multiple technologies and products.

From Associate, entry-level courses to Expert-level, experience-based exams, all professionals in or looking to begin a career in IT benefit from industry-leading training and certification paths from one of the world's most trusted technology partners.

Proven Professional certifications include:

- Cloud
- Converged/Hyperconverged Infrastructure
- Data Protection
- Data Science
- Networking
- Security
- Servers
- Storage
- Enterprise Architect

Courses are offered to meet different learning styles and schedules, including self-paced On Demand, remote-based Virtual Instructor-Led and in-person Classrooms.

Whether you are an experienced IT professional or just getting started, Dell Technologies Proven Professional certifications are designed to clearly signal proficiency to colleagues and employers.

Learn more at www.dell.com/certification

Table of Contents

Disclaimer: The views, processes or methodologies published in this article are those of the authors. They do not necessarily reflect Dell Technologies' views, processes or methodologies.

Abstract

Present computational models need to be revamped in order to meet the spike of context-aware content consumed as a service by urban inhabitants. The futuristic internet applications are embedded with AI, ML or VR/AR (virtual/augmented reality). These applications demand high reliability along with low latency for content delivery (AR, VR, MR) in real time provided by cloud service providers (CSP) with infinite resources in collaboration with content delivery networks (CDN) in various availability zones. CSP and CDN providers fall short in the existing hardware that works on 4G and LTE-powered networks supported by ICT and Telecom providers that are yet to roll out 5G in later phases.

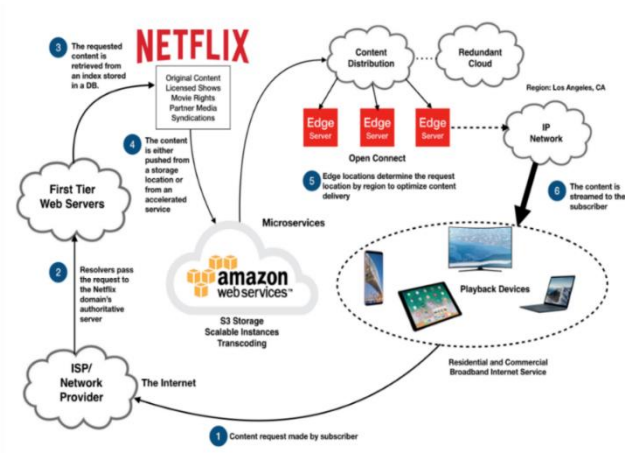
The emergence of new Internet of Things (IoT) applications across various verticals in implementation of digital cities brings us the context of Edge Computing in Urban populated areas. By placing part of cloud resources at the edge of the network, near the data sources and applications has created a new model known as **Urban Edge Computing**. This computing model has emerged as an extension to the cloud to support low latency and high-performance applications. The goal of Urban Edge Computing is to provide abstraction in a local setting of an urban area like any public, private or hybrid cloud computing. Like cloud, Urban Edge Computing must also solve a few challenges to achieve general acceptance. One such challenge is to set-up and continuously configure Edge Computing applications in an urban area where management overhead is required.

Presently, this problem has been addressed using a new paradigm called serverless technology in the cloud space. The concept of Serverless Computing at the edge is still in its infancy. In this article, we analyze function execution times in cloudlets, edge and micro datacenter. We also explore integration support and scalability cost constraints associated with serverless services provided by various vendors: Google Cloud Functions, AWS Lambda@Edge, AWS Greengrass, edjx.io and Azure Functions. We present implications of serverless computing on software pipelines in startup companies who are looking to run services in edge computing platforms in urban environment. We identify issues and research directions in this area of interest.

Introduction to Content Delivery Networks and Platforms

Content Delivery Networks (CDN) vendors pioneered the business models that act as an intermediary channel to deliver digital content services for web, mobile and legacy apps on the internet. For instance, online subscription models for pure-play online company survival and success depend on the quality of services provided by their CDN channel partner and contracting cost that deliver subscribed services to end-user devices.

Online gaming, social media and entertainment companies such as Netflix and Google (YouTube) provide their content across the heterogeneous devices used by theglobal urban population. Explosive growth in mobile user demands on digital content delivery – especially online video, pictures, and multimedia text and graphics – places a strain on network operator's capacity. A real-time use case which has forced every single telco service provider globally to build a CDN is the trending demand from many parts of the world to release blockbuster films or web-series. These could be more effectively streamed from servers close to requestors than struggling to scale from one point.



Open Connect Appliance

Netflix for instance uses more than one technology, using Open Connect Appliances (OCA's), fast compute and storage servers mostly assembled from lots of hard disks and flash drives to store videos. As far as hardware is concerned, there is nothing special in OCA. They are based on commodity PC components and assembled in custom cases by various suppliers, including Dell EMC.

Two new technologies – Serverless and Edge Computing – drove CDN vendors, Cloud Service Providers (CSP) and Software Defined Data Center solution providers like HP, Nutanix and Dell Technologies to not only venture and supply compute, network and storage hardware, but also cater software that is embedded in edge compute sensing solutions that are considered the next wave of popular disruptive forces. These disruptions continuously fortify demand for a pure "pay-per-use" model backed by a highly scalable platform for smart applications on the virtual infrastructure.

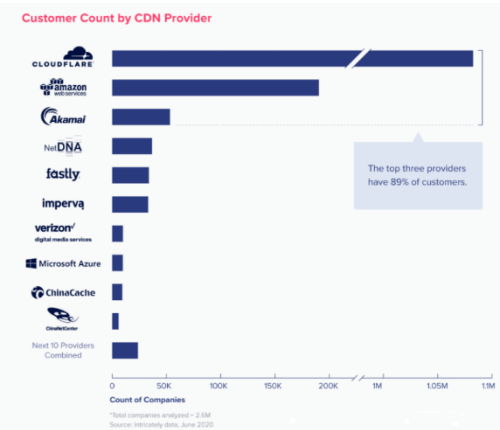
In this article, we elaborate on six popular CDN platforms that allow developing serverless functions on their edge nodes. We also shed light on the most widely used patterns in serverless applications design from a practitioner point of view. In addition, all these patterns have been further classified into five categories 1) orchestration and aggregation, 2) event management, 3) availability, 4) communication, and 5) authorization. We also propose the reasons why the software community developing edge computing solutions had been hesitant to promote Serverless. Because its inception was with CSP providers FaaS (Function as a Service) services with inherent nature of cloud-driven design of the current serverless platforms in combination with distinctive characteristics of the edge landscape and IoT applications.

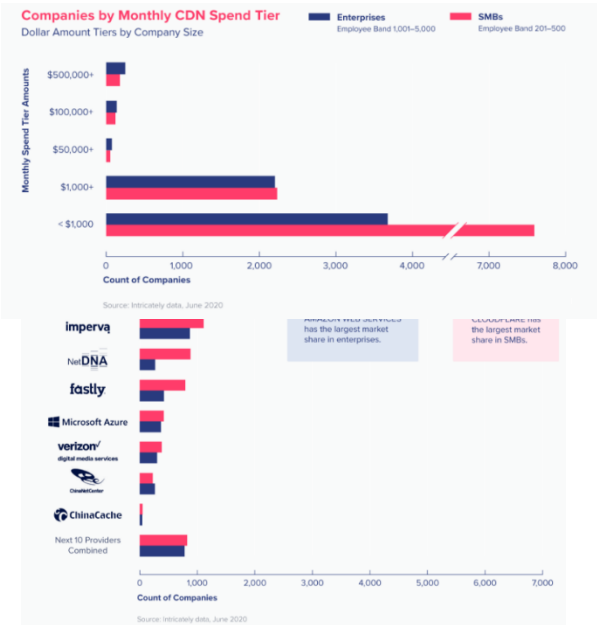
Finally, we use a real-world dataset in an open-source cloud environment based on experimental implementation of Knative showcasing that a serverless approach to manage IoT traffic is feasible. It concludes that a serverless approach uses less resources than a serverful approach. The prefetching mechanism can mitigate the cold start delay penalty suffered by serverless solutions, if we can analyze, anticipate and act on the traffic prediction using historical data as reference anchoring point.

Even before the global pandemic left its mark on global digital markets, the traditional CDN industry were on a pace of linear growth. Furthermore, COVID-19 sent tremors on the current CDN network providers to enhance their overlay network edge nodes to cater to accelerated demand for content delivery and venture into low latency edge-computing AR/VR sensing solutions.

With extended lockdowns, government agencies across most nations have enforced and encouraged urban populated areas to stay home and that schools provide children at-home digital content for e-learning. This has spiked the use of CDN for digital applications. Nielsen agency reported below:

Akamai has been at least a decade ahead in CDN business compared to most of its competitors. In





terms of revenue it is significantly ahead with respect to other CDN vendors. Most of its present revenue comes from its large-scale enterprise partners. While Cloudflare and AWS CloudFront have the most customers by count this value doesn't tell the whole story.

Category	Latency Group	Latency, ms		Application Examples	Infrastructural Solutions
		Boundary	Optimal		
I	Motion-to-Photon (MTP)	< 20	≈ 2.5	AR/VR, head-up displays	5G MEC
II	Perceivable Latency (PL)	< 100	≈ 40	Gaming, Video streaming	Cloudlets,
III	Human Reaction Time (HTR)	< 250	≈ 200	Remote surgery, Teleoperated machinery	Edge Clouds.
IV	Not critical for upstream aggregation, may vary for actuators or robotic control.			Miscellaneous IoT: smart cities and homes, electrical grids, manufacturing, agriculture, warehousing, etc.	Fog, Mist Clouds, etc.

Table : Edge applications, latency requirements, and infrastructural solutions.

Almost half of the Fortune Global 500 companies use Akamai. In October 2019, it set a CDN traffic record with 106 Tbps during the rollout of a Fortnite Chapter 2 update. Moreover, Akamai has the most enterprise level mid-market customers amongst Cloudflare, AWS, and Akamai. Traditionally, mid-market companies and enterprises spend almost $1,000 a month on CDN products. Note that less than 700 companies spend greater than $100,000 monthly on a CDN.

Mostly companies use multiple vendors for CDN requirements. However, AWS in USA and AlibabaCloud in China have been traditional leaders in cloud-CDN offerings. Furthermore, their approach with CDNs differs from other vendors in the market. Both offer affordable CDN infrastructure as part of their higher-yield services for next generation startup companies. Almost one-third of AWS customers are also Amazon CloudFront customers. In contrast, Azure CDN customer are fewer than 10% of Microsoft Azure customers.

Placing part of certain resources (e.g. compute, storage, logic) closer to the edge of the network enables faster and more context-dependent data analysis and storage solutions benefiting low-latency, real-time video surveillance and sensing solutions addressed through Edge Computing. (commonly termed Cloudlet for CSP vendors or MDC (Micro Data Center) for Telco or Smart City providers);

Edge Computing can be broken down as a set of nodes, each supporting different compute, storage, and network requirements. In today's market there are different flavors of Edge Computing networks alike the services provided by cloud vendors. For instance, on one-hand, a single organization looks after Private Edge Computing which consists of a private network of Edge Computing nodes. On the other hand, customers can deploy their services on top of a managed infrastructure known as Public Edge Computing, and we have a combination of the earlier two types known as Hybrid Edge Computing. We classified the Edge Computing requirements based on below four categories with respective to infrastructural solution.

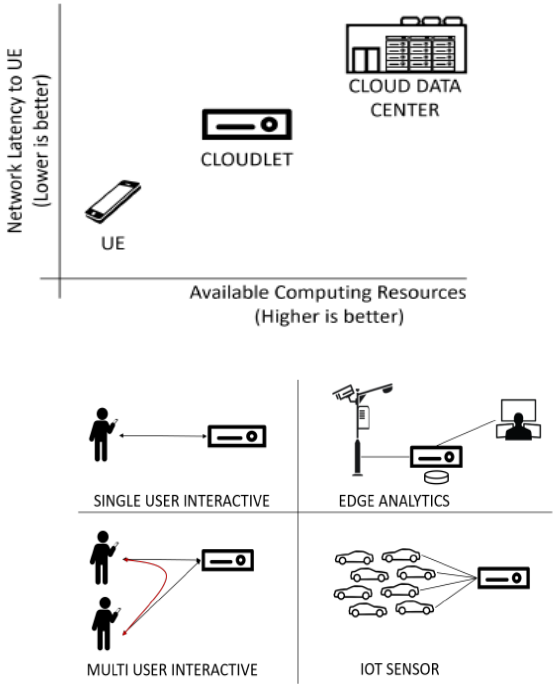
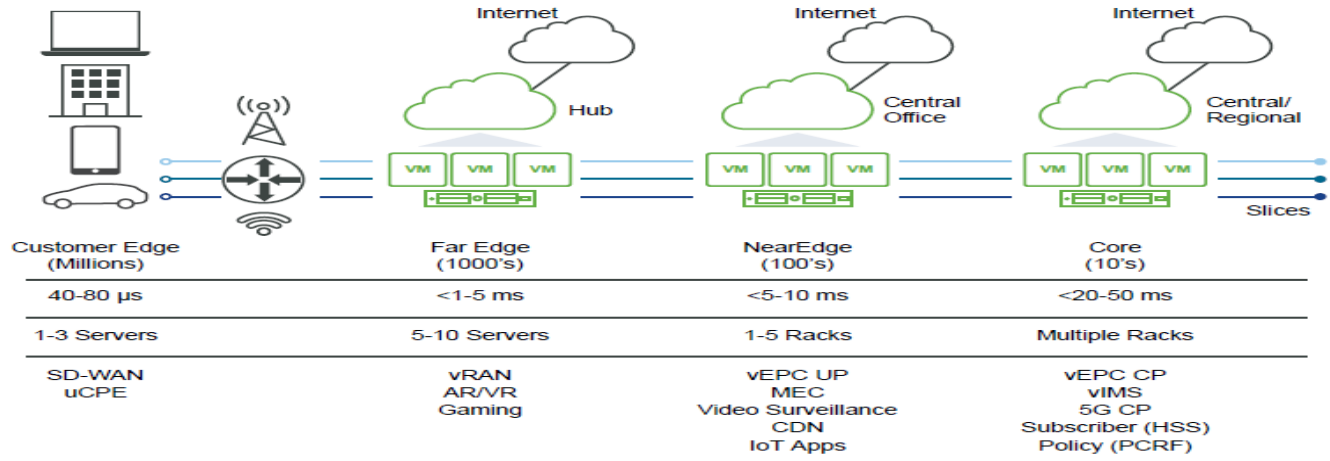


Figure . : Edge-native Application Categories

Figure : Mobile Distributed Application Architecture

Figure . Reference Environment



Based on the generic edge computing application concepts of offloading data or making use of computation resources at the edge where industry experts have encouraged data-centric perspective on how data is transformed and processed.

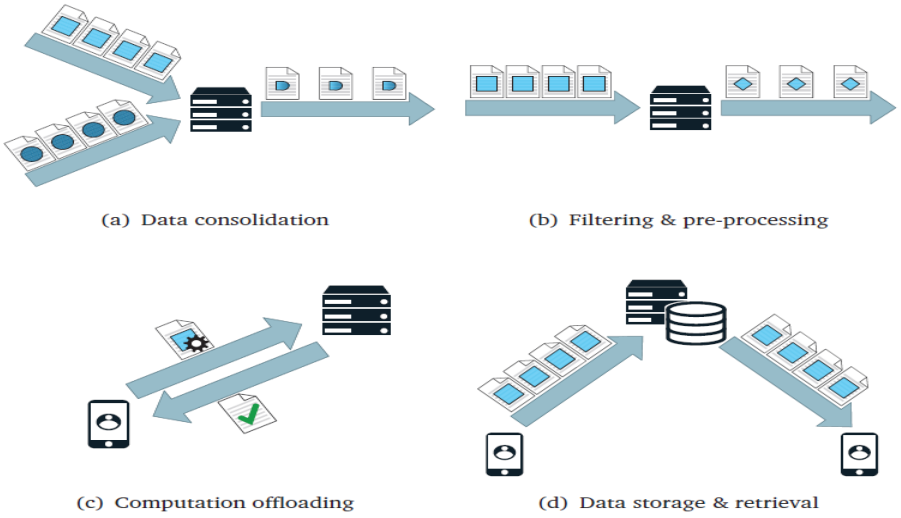


FIGURE : APPLICATION COMPONENTS†

We came up with the following four components used inside most latency-sensitive mission critical applications. Practitioners use all or a few components amongst them: (1) data consolidation (2) filtering & pre-processing (3) data storage & retrieval and (4) computation offloading. While the earlier two describe how data is transformed and are concerned on the flow of data, the last two indicate what happens with the data. Advancement in Telco clouds have further increased the computing nature for CSP, CDN and ISP providers for smart application usage.

We summarize the comprehensive list of the prominent use cases defined from Systematic Literature Review (SLR) where applications have benefited from edge computing.

For this, we use the following semantics:

Edge Computing is vital to ensure the requirements are met, and these cannot be fulfilled by cloud itself. In addition, processing at end-user devices cannot ensure the expected quality of experience. Edge Computing improves the quality of the service and/or its experience for the users. The advantages of Edge Computing usage depend on the context in which the application operates. Edge Computing brings no real-world advantage, and the attribute is irrelevant for the application. For instance, this might not be critical in applications where computation or actuation takes far longer than the communication. Even so, Edge Computing might improve latency in absolute numbers. The last four columns of the table indicate which of the defined components are used in the use cases.

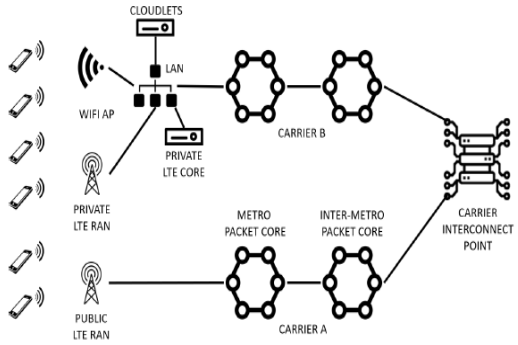


Figure : Framework Physical Architecture

	Latency	Bandwidth	Energy	Privacy	Consolidation	Filtering	Storage	Offloading
Environment								
Pollution monitoring [ZLH13; Zhe+14b]	–	○	–	–	✓	✓	✓	✗
Pollution monitoring via crowdsensing [Mai+09] [Sch+12; Dut+17; MAŽ18]	–	+	+	+	✓	✓	✓	✗
Optimizing garbage collection [AZM15; Per+14b]; [Aaz+16; Med+15]	–	–	–	–	✓	✓	✓	✗
Emergency Response								
Emergency notification [AH15]	–	○	–	–	✗	✓	✗	✗
Situation awareness/ mobile command and control [Chu+13]	○	+	–	○	✓	✓	✓	✗
Ad-hoc communication in disaster scenarios [Meu+17c] [Sat+13]	+ +	+ +	–	–	✓	✓	✓	✗
Surveillance								
Vehicle tracking [Che+17a]	○	+	–	+	✗	✓	✗	✓
Just-in-time video indexing [Sat+17]	+	+ +	○	+	✗	✓	✓	✓
Biometric identification [Sto12]	–	○	○	+	✗	✗	✓	✓
IoT Device Augmentation								
Smart Home/Building								
Video surveillance [San+14] [AXS17]	–	+ +	–	+	✗	✓	✓	✗
Coordination of building subsystems [Fer+18]	○	○	–	+	✓	✓	✗	✗
Industrial IoT								
Production process analysis [AZH18; LGS17; Fu+18]	–	○	○	+	✓	✓	✓	✗
Machine condition monitoring [Wu+17a; Oye17]	○	+	–	+	✓	✓	✗	✗
Warehouse logistics scheduling [LY18]	○	+	–	+	✓	✓	✗	✗
Dynamic production line scheduling [Wan+18a]	+	○	–	+	✓	✗	✗	✓
Agriculture & Farming								
Monitoring plants/lifestock [ADH18; Car+17]	–	+ +	+	○	✓	✓	✗	✗
Yield prediction [Lia+18]	–	○	–	+	✓	✗	✓	✓

	Latency	Bandwidth	Energy	Privacy	Consolidation	Filtering	Storage	Offloading
Mobile Device Augmentation								
Gaming								
Scene rendering [MKB18b; LS17]	+ +	+	+	–	✗	✗	✗	✓
Collaboration of neighboring players [Cai+18; PS18]	+ +	+	+	–	✓	✗	✓	✓
AR/VR								
Rendering [Shi+19]	+ +	+ +	+	–	✗	✓	✗	✓
Hybrid rendering [Lai+17]	+ +	+	+	–	✗	✓	✓	✓
Reconstruction of 3D maps [Boh+15]	+	+ +	+	–	✓	✓	✓	✓
Content delivery								
Video streams [AD14]	–	+	–	–	✗	✓	✓	✗
Website delivery [Zhu+13]	–	○	–	–	✗	✗	✓	✗
Applications and updates [Bha+15b; Bha+15a]	–	+	–	–	✗	✗	✓	✗
Collaborative caching [LYS16; Tra+17]	+	–	○	○	✗	✓	✗	✗
Storage								
Storage for edge analytics [LMB17]	+	+	○	○	✓	✓	✓	✓
Reverse CDN [Sch+17; Ged+18b] [Psa+18; MSM17]	○	+	○	+	✗	✗	✓	✗
Document synchronization [Hao+17]	–	+	–	○	✓	✗	✓	✗
Personal data storage [Cha+15; Mor+16; Per+17b]	○	○	○	+	✗	✗	✓	✗
Infrastructure Augmentation								
Smart Grids								
Monitoring and control [Fre+13]	○	+	○	+	✓	✓	✓	✗
Scheduling distributed energy resources [PSM10]	+	○	–	○	✓	✗	✗	✓
Traffic & Transportation								
Adaptive traffic light [Gha+16]	–	○	–	+	✓	✓	✗	✓
Detection of road hazards [CDO19]	○	○	–	–	✓	✓	✗	✓
Traffic planning [Zhe+11; Sat+17; QKB17]	–	○	–	○	✓	✓	✓	✗
Emergency vehicle route clearance [Nun+15]	–	+	–	–	✓	✗	✗	✗
Autonomous Driving								
Disseminating data to vehicle [DBH15; Yua+18]	+ +	+	–	○	✓	✓	✗	✗
Processing LIDAR data [Qiu+18]	+	+ +	–	–	✗	✓	✗	✓

	Latency	Bandwidth	Energy	Privacy	Consolidation	Filtering	Storage	Offloading
Human Augmentation								
Quantified Self								
Analyzing fitness tracker data [Sch+15; Baj+15]	–	○	+	+	✗	✓	✓	✓
Precision Medicine								
Fall detection [Cao+15]	–	+	+	+	✗	✓	✗	✓
Patient monitoring with WBAN [AS16]	○	+	○	+	✓	✓	✓	✓
Remote surgery [Xu+14]	+	+	–	○	✗	✓	✗	✗
Analyzing ECG features [AG10]	–	○	–	+	✗	✓	✓	✓
Cognitive Assistance								
Face recognition [RCR11]	+	+	○	+	✗	✗	✗	✓
Speech recognition [Agu+10]	+ +	○	○	+	✗	✗	✗	✓
Wearable cognitive assistance [Ha+14]	+ +	+	+	+	✓	✓	✗	✓

Table . Comparison of the CDN platforms

	CDN Platforms with Serverless support						
Features	**Akamai**	**Cloudflare**	**Stackpath**	**CloudFront**	**Edjx**	**IBM Edge Functions**	**Nuclio**
Support of AI on the edge				✓	✓		✓
Availability	Globally	Globally	Limited	Globally	Limited	Globally	Limited
Supported platforms (edge hardware)	Akamai nodes	Cloudflare nodes	Stackpath nodes	AWS nodes	nanoservers	IBM centers	Portable across constrained devices
Supported Languages	JavaScript	JavaScript	multiple languages	multiple languages	multiple languages	JavaScript	multiple languages
Cost model	Pay as you go						hosting cost
License	Proprietary						Open source

As per our knowledge and experience, CDN solution providers are targeting AR/VR applications along with the Gaming and Video streaming industry. Emergence of serverless solutions by Content Delivery Network platforms take advantage of emerging markets to offer a pay-per use model.

They mathematically calculated the CapEx vs OpEx expenditure to not only check the potential benefits of providing serverless support in their nodes by offering caching of the web content on their nodes, but also, to take advantage of computational capabilities in their present infrastructure nodes using the serverless technology. We would discuss the contrast of six CDN platforms that presently offer developing serverless functions on their edge nodes (Table). Presently, CloudFront, Edjx and Nuclio are the only platforms that support AI on the edge.

Table . Comparison of the IoT platforms

	IoT platforms with serverless support				
Features	**AWS GreenGrass**	**Azure IoT**	**FogFlow**	**OpenWhisk-Light**	**Nuclio**
Support of AI on the edge	✓	✓	✓	✓	✓
Availability	Globally	Globally	Limited	Limited	Limited
Supported platforms (edge hardware)	Hardware supporting docker Containers	Tier 1: Hardware supporting containers. Tier 2: Hardware supporting virtual machines.	hardware supporting docker containers.	Multiple container frameworks. Demonstrated for limited operation also in Raspberry Pi	Portable across constrained devices
Supported Languages	multiple languages	multiple languages	multiple languages	multiple languages	multiple languages
Cost model	Pay as you go		Private setting		private or hosting cost
License	Proprietary	Open source			

Presently, various solutions have been proposed on the market for IoT Serverless on the edge. Moreover, serverless is not yet mature. On the one hand, existing Edge providers are extending Existing edge platforms these can deploy functions written in constrained set of languages. While new platforms enable developers to use different languages. In-contrast the traditional CDN platforms restrict developers to write Javascript code on their edge node.

An interesting premise that came out of this industry research is that several CDN providers offer edge support to enhance the performance of web systems. They are exploring the possibility to deploy code as serverless functions that would enable dynamic web pages to be composed on the edge running a part of the business logic for AR/VR content delivery solutions in the gaming industry.

Patterns for Serverless used by IoT vendors and CDN Solution providers

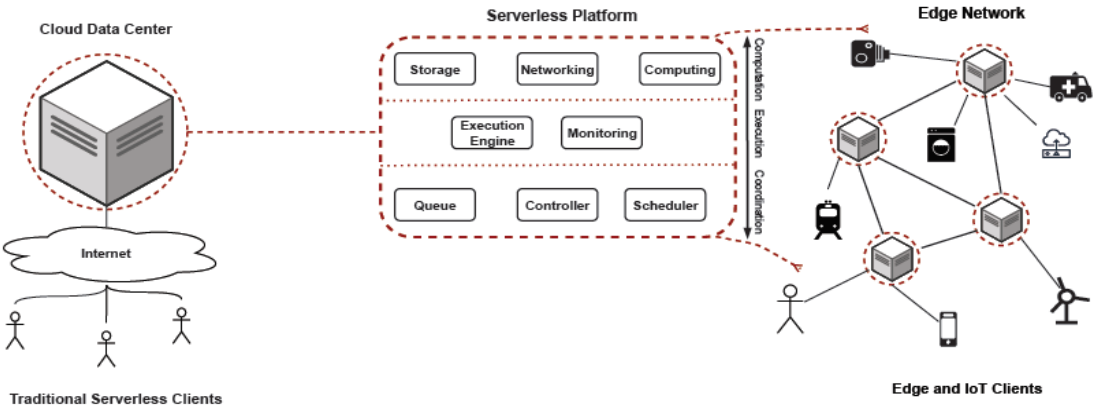


Figure : A simplified Serverless Edge Computing Paradigm.

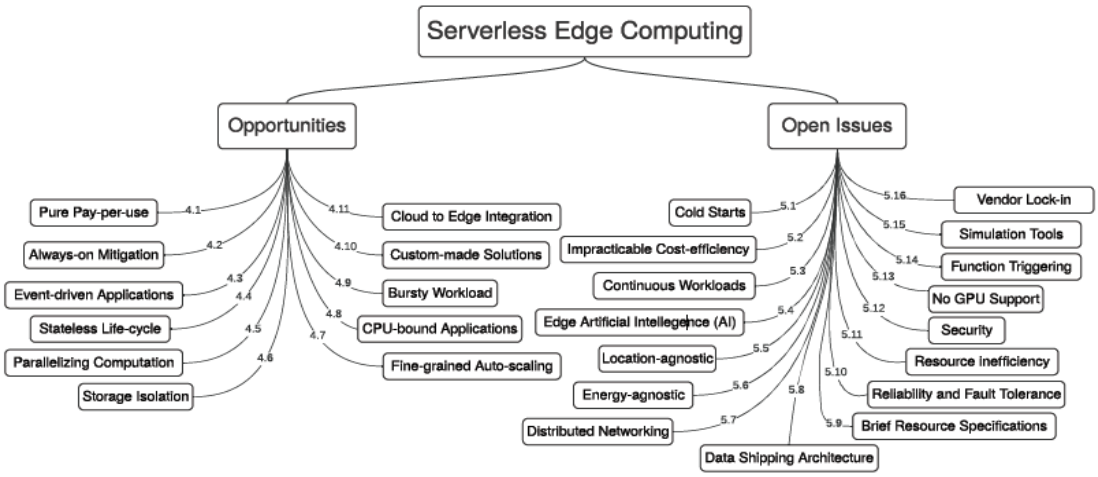


Figure : A Taxonomy of identified opportunities and open issues for Serverless Edge Computing.

The figure above lists opportunities and open issues for serverless edge computing. Industry practitioners are developing next-level augmented reality (AR)- and virtual reality (VR)-related gaming solutions that aim to solve open issues such as cold-starts and vendor lock-in. In particular, focus has been given to resolve issues pertaining to Edge AI solutions that overlap between CDN and Cloud Providers.

Early adoption of 5G networks in South Korea and their impact on mobile VR experience is a prime example of where ultra-low latency is an absolute requirement for gaming production studios.

Edge Solution Offerings

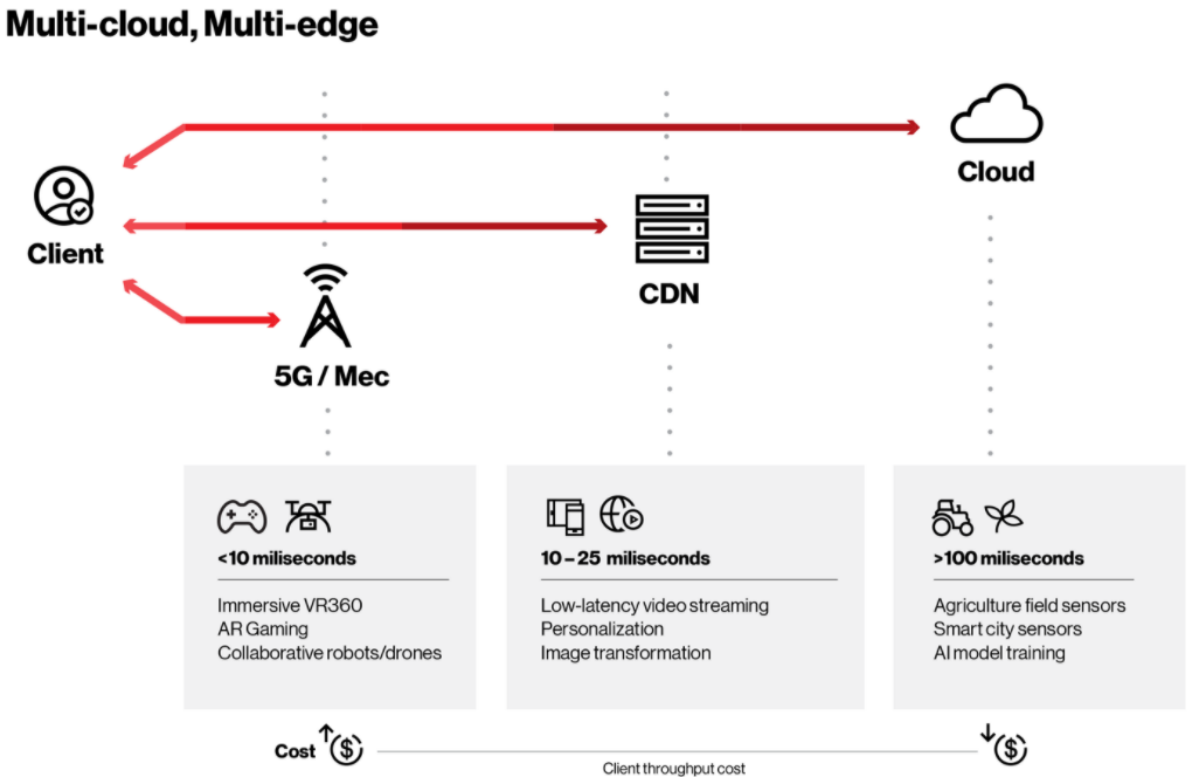
Edge computing is so complex that one needs to choose the right edge from multiple edge solution offerings. The categories of multiple edges are shown below:

1) Device Edge

2) 5G Edge

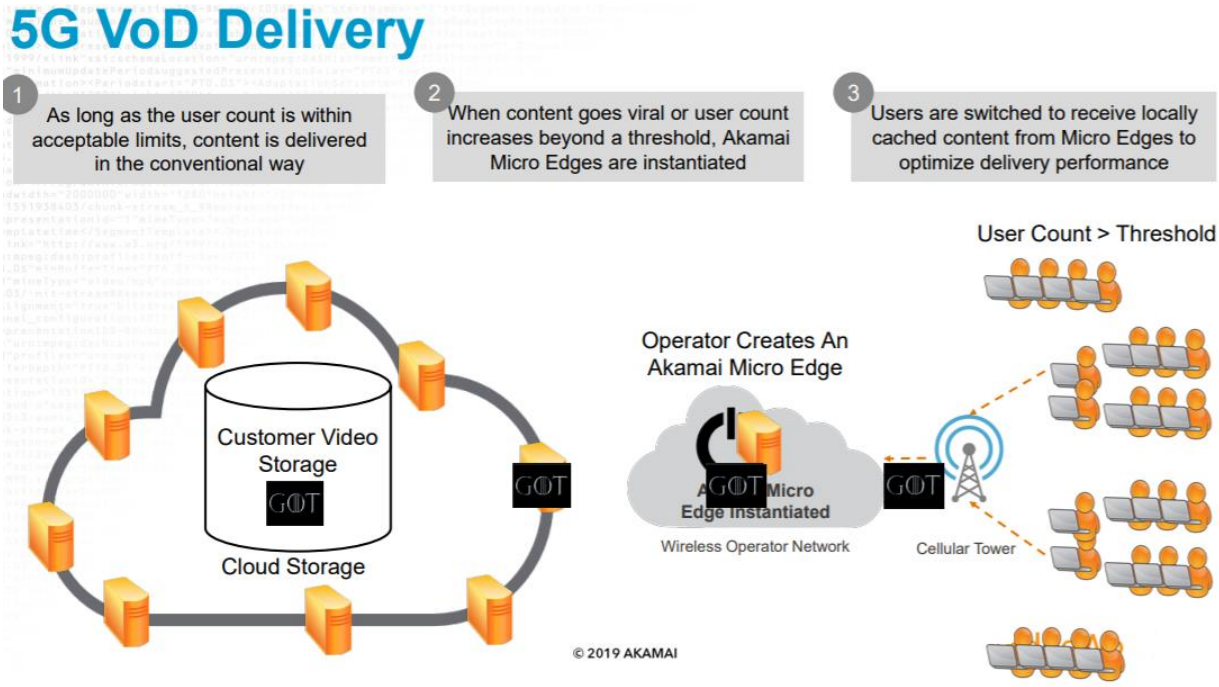
3) CDN Edge

4) The Cloud



We are sticking to CDN edge in this article where there are certain serverless patterns used to cater evolving needs in the smart application service-based industry. CDN Edge is the only edge solution that offers more compute capability than the 5G Edge/MEC (Mobile Edge Computing) and supports high bandwidth practically. However, CDN Edge increases the latency ranging from >=10ms as of now. Presently gaming companies and VR/AR production houses are moving their applications and workloads from the Cloud to the CDN Edge for the cost benefit offered from pay-per use model that not only offers lower latency but also brings a better experience to their customers. Click here to learn more about the CDN Edge offered by Verizon Media Platform that has more than 5,000 last mile networks, providing global scale, performance, and speed. The Verizon network presently offers 100+ Tbps of egress capacity altogether.

Serverless Design Solutions



For our work on serverless and CDN Edge, we chose and referred to GL (Gray Literature) which refers to many forms of organizational reports – most are publicly available including blogs and online articles from industry and governments. We identified 32 patterns for serverless design solutions from the Gray Literature and consolidated into these five categories:

1) orchestration and aggregation

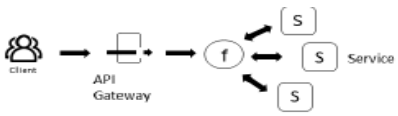
2) event management

3) availability

4) communication

5) authorization

Orchestration and Aggregation: Practitioners face a conundrum in server vs. serverless orchestrating execution requirements while resolving complex functions or microservices in multiple edges of core, edge and cloud. The developers utilize the below pattern to compose serverless functions mostly known as aggregators; They are mostly known as "Durable functions". (Aggregator Pattern) Problem: Several APIs are exposed in a single endpoint.



Solution: Aggregating the results and exposing them as a singular endpoint when a function calls APIs before sharing to end-result to client. Mostly an API Gateway is used in these scenarios before the desired function.

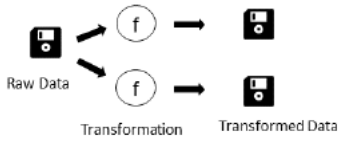
Data Lake

Problem: Evolving hassles to process large scale real-time data and performing transformations on assigned data is difficult to resolve.

Solution: Physical storage of raw data where data is processed and deleted is known as data lake. The least likelihood of organizing metadata in sensible manner with naming convention to keep order with changing times.

Benefits: The data can be transformed anytime, independent from the needs of the present moment;

In-addition, the data remains identical every time.

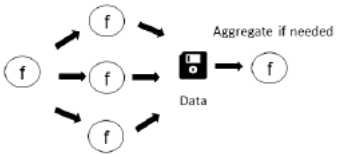


Fan-In/Fan-Out

In Gray Literature this scenario is known as “Virtual Actors”, “Data transformation”, “Processor”, “Fire triggers and transformations”.

Problem: Like Function Chain, usually long tasks exceed the maximum execution time to enable an execution

Solution: The simultaneous execution of functions leads to faster results; the aggregated results in the end are the result of the divided work processes in executed in parallel.

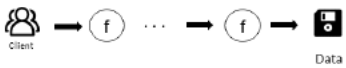


Issues: Chained functions are cohesively joined. Splitting the tasks between functions can be complex in a function chain.

Function chain

Problem: Executing long tasks that exceed the maximum execution time are enabled through Function Chain. For instance, running lambda longer than 15 minutes.

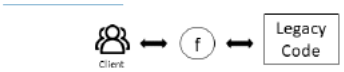
Solution: The initial running functions are terminated without affecting the next function in the chain by passing each parameter asynchronously to those functions that are needed to continue the computation without breaching the maximum execution time. A chain of functions is combined, while keeping track of the remaining execution time. When the computation of initial function is kickstarted.



Issues: Splitting the tasks between functions would increase complexity when there is drastic increase in number of functions is found when there is heavy coupling between chained functions.

Proxy: In GL literature, this pattern is known as “Command pattern”, “Anti-Corruption Layer”;

Problem: Legacy system is supposed to be integrated with functions written in various languages.



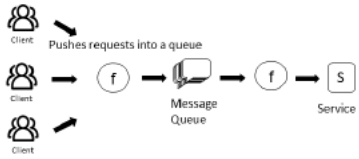
Solution: A proxy for another service is created through a function that acts as a translator for the requested data format those are necessary for any requested protocol.

Benefit: Simple to access API for clients.

Queue-Based Load Leveling

This pattern is popularly known as "The Scalable Webhook", "The Throttler" in GL Literature.

Problem: With non-scalable back-ends, we are trying to build scalable webhooks. With custom callbacks the Webhooks usually augment the web application or alter the behavior of a web page.



Solution: Under heavy load, when Queue services are used to trigger a function. The frugal consumer enables requests to be queued.

The Frugal Consumer

Problem: Non-scalable backends are supposed to increase scalability.

Solution: Post the messages directly to a message queue. Helpful when a function is supposed to process requests of multiple services.

The Internal API

Problem: Cloud infrastructure services that are used within the boundary of certain CSP vendor are typically built on microservices framework for access control.

Solution: Invoking the functions through HTTP using certain Invocation Type.

Benefits: They s are not accessible from outside the boundary; Leveling up the security as service.

The Robust API

In GL literature it is commonly known as "The Gateway

Problem: The services that provide accesses in the backend are known to the client in few scenarios.

Solution: The mediation is done through API Gateway for selected services that are accessible for clients through required policy and grants.

Benefits: Individual clients and can be handled effectively.

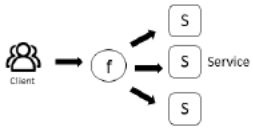
Issues: Complexity increases if not done appropriately.

The Router

Widely called “Routing Function”, “Decoupled Messaging”, “Data probing”

Problem: Avoiding additional cost payments that occur in orchestration systems that are implemented in the state machine pattern. The executions are distributed mostly based on payload.

Solution: Based on the incoming request payload the related functions are invoked. Usually, a function that acts as a router is created.



Benefit: Simplifies implementation.

Issues: The routing function needs to wait until the target function terminates the execution which brings in the double billing issue. Moreover, performance bottlenecks and single point of failure are introduced through implementation of routing functions.

Thick Client

Problem: Increase in costs and latency with the introduction of intermediary layer between client and service.

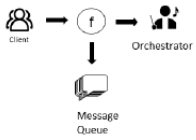
Solution: Services and orchestrated workflows are handily accessed by clients.

Benefits: Improvemes separation of concerns, better performance and economical cost at server level.

The State Machine

Problem: To achieve the desired state, the functions must coordinate and orchestrate amongst themselves.

Solution: AWS Step Functions, Azure Durable Function, AWS s3, or IBM Composer complex tasks are performed through orchestration. These serverless orchestration are being adopted widely.



Issue: Development time and effort grow tremendously with increased complexity in the system.

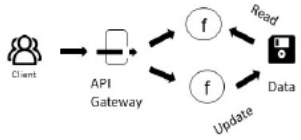
Event-Management

Classified patterns falling under the managing events category; it mostly caters to resolve communication problems.

Responsibility Segregation

Problem: Increased use of the same functions for queries and data updates can result in rigidity.

Solution: The preferred approach is to segregate functions at data sources where they are read and updated distinctly. To avoid this congestion, one must use ”Commands and Queries” through an appropriate function.



Distributed Trigger

Commonly known as “EventBroadcast” in serverless platforms.

Problem: Only with its own service, a message queue topic is coupled as shown below.

Solution: The above setup works well with micro services where the data confined to the boundary of required microservices and the topics are used for single purpose.

Issues: The individual services are responsible for the subscriptions of the queue or topic.

FIFO

Problem: FIFO approach (first in, first out) does not solve issues all the time. We have to create FIFO Queue for desired end state in serverless functions

Solution: To periodically invoke the function asynchronously, one has to use AWS Cloudwatch and set concurrency to 1. This disables competing requests to run in parallel. For instance, <= 10 messages are polled through the functions before processing is done. The function removes the messages from the queue when processing is complete, and later invokes itself again (asynchronously). The process repeats itself until all the items have been removed from the queue.

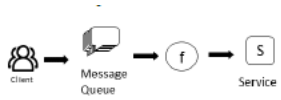
Benefits: Sequencing is simplified.

Issues: The retry will continue to cascade If the self-invocation is blocked. The cronjob will fail because of the concurrency settings while the function is engaged in processing messages.

The Internal Hand-off

Problem: Processing asynchronous event while using invocation Type (Event)

Solution: When enabling Dead Letter Queue to capture failures, one is required to use a message queue.



Periodic Invoker

Problem: Tasks are executed in a certain manner within a stipulated period.

Solution: AWS Cloud Watch, Google Cloud Scheduler, or Azure Scheduler are used to subscribe to a function that has been scheduled.

Benefits: Without the need to keep them alive all the time, one can run functions periodically.

Polling Event Processor

Known in GL (Gray Literature) as "Polling consumer".

Problem: External systems that do not publish events when there has been change in state.

Solution: To check the state of the service, one needs to use the Periodic Invoker pattern.

Benefits: Without Listener being enabled to keep a function alive permanently, functions must be run periodically.

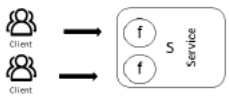
Availability Patterns

To solve availability problems such as reducing possible failures during the warm-up time, the patterns below are used.

Bulkhead

Problem: The complete system risks being compromised when crucial functions catering heavy load fails.

Solution: Based on workloads, various pools will be created, forcing practitioners to partition workloads into different pools.

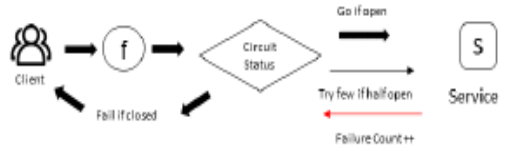


Benefits: To reduce failure risks caused by cascading chain reaction, one needs to isolate failure in processes.

Circuit breaker

Problem: All low performing API calls are monitored and failed API's are tracked.

Solution: Based on certain threshold being breached "open" the circuit sends errors back to the calling client instantly without even trying to call the API. After a brief timeout, the system will "half open" the circuit and send a few requests to check if the API is responding correctly. If the sample requests are successful, the system moves to "close" the circuit and lets all the traffic pass through it. However, if any of the requests receive an error again, "open" circuit come into the picture.



Benefits: This pattern has been found useful to cater cost benefits for synchronous requests,.

Compiled Functions

Problem: If the desired computing invocation time had not been so heavyweight requiring in-memory footprint, Serverless cloud computing would have been a perfect fit for IoT solutions giving the practitioner a choice of choosing amongst the multiple edges.

Solution: To make Edge technology viable in the cloud, various Serverless languages have been introduced to reduce the invocation time and memory footprint.

Function warmer

Commonly referred as" Function Pinging", "Warmer service", "Cold Start", "Keeping Functions Warm", "keep-alive" in the Gray Literature.

Problem: After a function is invoked, there is a delay found before the function starts to be executed. Referred as cold start time, It usually takes between 1 and 3 seconds across various vendors. The functions are executed in containers that encapsulate and execute the desired code in serverless environments. Once a new function is invoked, after the execution of the function they code is considered warm. The request would be served instantaneously if the container continues running only for a certain time period. However, before the shutdown if another request comes in, recycle idle function instances are introduced in AWS and Azure after a fixed period of 10 and 20 minutes respectively.

Solution: To keep the function warm, one needs to ping the function periodically.

Benefit: Response times can be decreased from 3 seconds to 200 milliseconds using the above method.

Issues: Even if we limit ourselves to make only one call every 10-15 minutes, we have noticed there has been steep increase in cost while applying this pattern.

Oversized Function

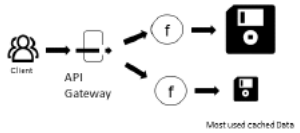
Problem: Presently it is not possible to choose functions to run on desired CPU's.

Solution: Even if extra memory is not the desired requirement, desired results can beachieved by provisioning virtual machines that cater bigger memory to run oversized functions.

Read-heavy report engine

Problem: Read-intensive applications are supposed to overcome downstream limits.

Solution: Most frequently queried data can be catered through creation of specialized views that uses data caches.



The Eventually Consistent

Problem: To keep data consistent amongst services, one needs to replicate data between services.

Solution: Trigger events based on changes done on the database based on earlier functions handled. For instance, use database stream services such as DynamoDB stream in AWS.

Timeout

Problem: Although API gateways suit many requirements, its idle timeout – 29 seconds – is long enough for users to consider it a bad experience for a given service in the gaming industry.

Solution: Most favored timeout is around 3-6 seconds.

Communication Patterns

To communicate between functions, the below set of patterns are used.

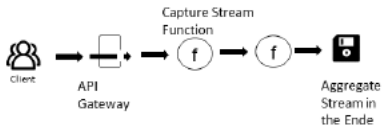
Data Streaming

Referred to in Gray Literature as" Stream and pipeline"," I am "a streamer" and "Event Processor", "Streaming Data Ingestion", "Stream processing".

Problem: Continuous stream of data are supposed to be managed.

Solution: For instance, Kinesis (AWS) contributes to handle and distribute large streams of data as a service.

Issues: In Serverless, working outside the proprietary platform's ecosystem would be difficult, should one choose to do so.



Externalized state

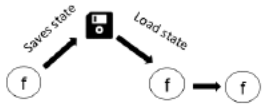
In Gray Literature, this is known as “Share State”.

Problem: In some scenarios, the state between the functions are to be exchanged.

Solution: External databases are used to share the state by saving it.

Issues: Higher latency overhead has been observed. Additional programming effort is required to resolve the cohesive coupling between the functions.

Solution: For instance, it is possible to use any number of services to pipe data to a Kinesis stream. Kinesis can be used to aggregate the results of the large volumes of events or data that are captured through Continuous stream processor that distributes them to different data stores as fast as they arrive. AWS API Gateway can be used as a Kinesis proxy.



Publish/Subscribe

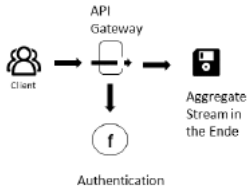
In Gray Literature this is called “The Notifier”:

Problem: Internal services (or APIs) are used to forward data.

Solution: To distribute internal notifications for internal services, one has to use standalone topic in the message queue.

Authorization Patterns

The pattern below deals with user authorization problems.



The Gatekeeper

Problem: Authorize functions.

Solution: In this scenario, use a Gateway to create an authorizer function that processes the authorization header and returns the authorization policy.

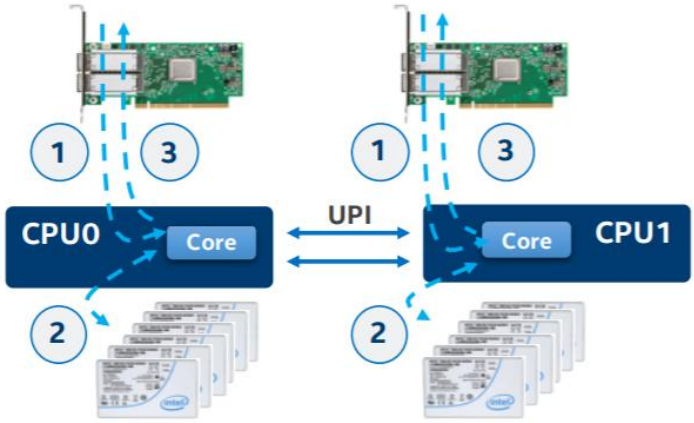
Valet key

Problem: Handling the authorization, without routing all the traffic through a gatekeeper process.

Solution: A token which is valid for a certain period and provides viable access rights. This serverless function grants access from a special authority by requesting it first.

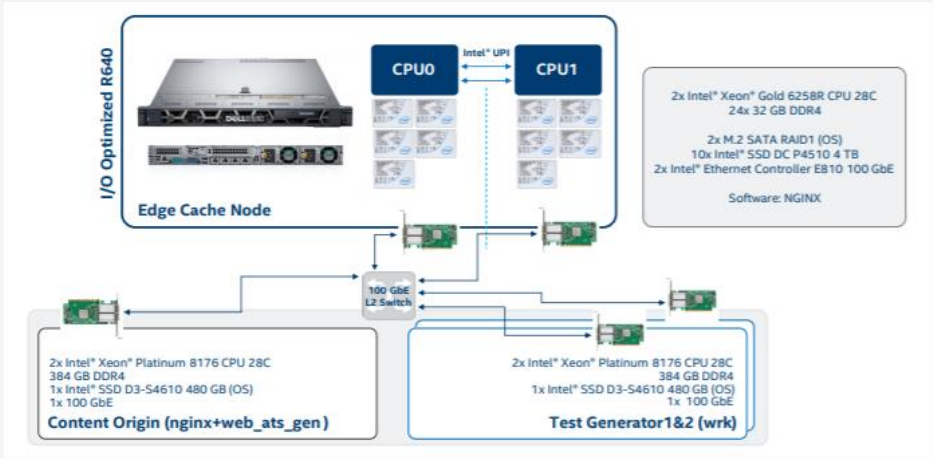
The time taken for the CPUs to communicate with each other and pass information can slow performance, impacting CDN quality of service. The percentage of communication between the CPUs is greatly reduced, resulting in higher overall performance. This was achieved through the present configuration of the I/O Optimized PowerEdge R640.

1. Request from Network
2. Read data from local storage into local memory
   Operate on data
3. Write data back out to network

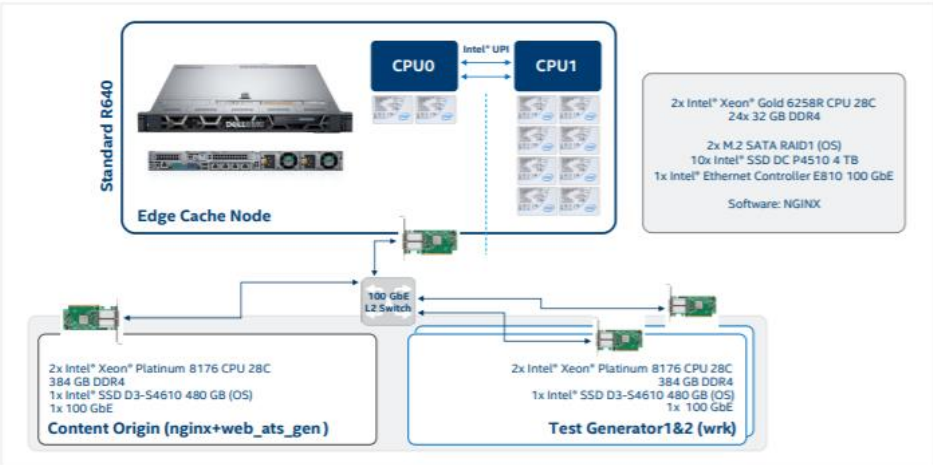


Only three steps are needed for an operation performed locally with balanced storage and network.

Solution Brief | New Dell I/O Optimized PowerEdge R640 Delivers CDN Performance



Servers used in CDN performance tests of I/O Optimized PowerEdge R640.



, Servers used in CDN performance tests of Standard PowerEdge R640.

Solution Brief | New Dell I/O Optimized PowerEdge R640 Delivers CDN Performance

	AVG. NETWORK THROUGHPUT (Gb/S)	WRK BANDWIDTH (GB/S)	AVERAGE LATENCY* (ms)	WORST CASE P99 LATENCY* (ms)
Standard R640 Config (bare metal software configuration)	99.40	11.47	54.30	456.61
I/O Optimized R640 (containerized)	182.90	21.06	30.77	334.17

Optimal Software Configuration NGINX CDN Performance[1]

	AVERAGE NETWORK THROUGHPUT (Gb/S)	WRK BANDWIDTH (GB/S)	AVERAGE LATENCY* (ms)
Standard R640 Configuration	87.93	10.08	54.85
I/O Optimized R640	182.90	21.06	30.77

Containerized NGINX CDN Performance[1]

The table above shows the results of the bare metal and containerized tests conducted on both the I/O Optimized PowerEdge R640 and the standard PowerEdge R640.

Conclusion

This article described the most common platforms and technology used behind both server-full and Serverless offerings in Cloud and CDN as well as enhanced services using Edge Computing to mobile users. Few of the selected platforms were targeted with concern towards IoT, as most of them are pursuing CDN. It was thought-provoking to notice that many popular CDN providers presented edge support to increase the performances of web systems. While edge computing in CDN is still in the nascent stage, its significance has been realized by those academicians and practitioners wish to explore work on the possibility to deploy code as serverless functions. Now, Enterprises and Streaming service application owners have the ability to not only compose dynamic web pages on the edge, but also run part of its business logic.

In forthcoming work, we plan to investigate and study those companies that adopted serverless and CDN solutions and benefited from these two technologies. We will also recommend when to use a mixture of the solutions or avoid them altogether.

References

[1] Paarijaat Aditya, Istemi Ekin Akkus, Andre Beck, Ruichuan Chen, Volker Hilt, Ivica Rimac, Klaus Satzke, and Manuel Stein. 2019. Will Serverless Computing Revolutionize NFV? Proc. IEEE 107, 4 (2019), 667–678.
[2] Zaid Al-Ali, Sepideh Goodarzy, Ethan Hunter, Sangtae Ha, Richard Han, Eric Keller, and Eric Rozner. 2018. Making serverless computing more serverless. In International Conference on Cloud Computing (CLOUD). IEEE, 456–459.
[3] Mohammad S. Aslanpour, Sukhpal Singh Gill, and Adel N. Toosi. 2020. Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research. Internet of Things 12 (2020), 100273. https://doi.org/10.1016/j.iot.2020.100273
[4] Javadi Bahman, Sun Jingtao, and Ranjan Rajiv. 2020. Serverless architecture for edge computing. In Edge Computing: Models, technologies and applications. Institution of Engineering and Technology, 249–264. https://doi.org/10.1049/pbpc033e_ch12
[5] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, and Aleksander Slominski. 2017. Serverless computing: Current trends and open problems. In Research Advances in Cloud Computing. Springer, 1–20.
[6] Luciano Baresi, Danilo Filgueira Mendonça, and Martin Garriga. 2017. Empowering low-latency applications through a serverless edge computing architecture. In Europ. Conf. on Service-Oriented and Cloud Computing. Springer, 196–210.
[7] Rajkumar Buyya, Satish Narayana Srirama, Giuliano Casale, Rodrigo Calheiros, Yogesh Simmhan, Blesson Varghese, Erol Gelenbe, Bahman Javadi, Luis Miguel Vaquero, and Marco A S Netto. 2018. A manifesto for future generation cloud computing: Research directions for the next decade. ACM computing surveys (CSUR) 51, 5 (2018), 1–38.
[8] Paul Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2019. The rise of serverless computing. Commun. ACM 62, 12 (2019), 44–54.
[9] Ryan Chard, Tyler J Skluzacek, Zhuozhao Li, Yadu Babuji, Anna Woodard, Ben Blaiszik, Steven Tuecke, Ian Foster, and Kyle Chard. 2019. Serverless supercomputing: High performance function as a service for science. arXiv preprint arXiv:1908.04907 (2019).
[10] Claudio Cicconetti, Marco Conti, Andrea Passarella, and Dario Sabella. 2020. Toward Distributed Computing Environments with Serverless Solutions in Edge Systems. IEEE Communications Magazine 58, 3 (2020), 40–46.
[11] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. 2020. Edge intelligence: the confluence of edge computing and artificial intelligence. IEEE Internet of Things Journal (2020).
[12] Jasenka Dizdarević, Francisco Carpio, Admela Jukan, and Xavi Masip-Bruin. 2019. A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration. ACM Computing Surveys (CSUR) 51, 6 (2019), 1–29.
[13] Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina Abad, and Alexandru Iosup. 2020. Serverless Applications: Why, When, and How? IEEE Software (2020).
[14] Simon Eismann, Joel Scheuner, Erwin van Eyk, Maximilian Schwinger, Johannes

Grohmann, Nikolas Herbst, Cristina L Abad, and Alexandru Iosup. 2020. A review of serverless use cases and their characteristics. arXiv preprint arXiv:2008.11110 (2020).
[15] Nabil El Ioini, David Hästbacka, Claus Pahl, and Davide Taibi. 2020. Platforms for Serverless at Edge: A Review. In 1st Int. Workshop on Edge Migration and Architecture (EdgeWays 2020).
[16] Paula Fraga-Lamas, Tiago M Fernández-Caramés, Manuel Suárez-Albela, Luis Castedo, and Miguel González-López. 2016. A review on internet of things for defense and public safety. Sensors 16, 10 (2016), 1644.
[17] Mostafa Ghobaei-Arani, Alireza Souri, and Ali A Rahmanian. 2020. Resource Management Approaches in Fog Computing: a Comprehensive Review. Journal of Grid Computing 18, 1 (2020), 1–42. https://doi.org/10.1007/s10723-019-09491-1
[18] Alex Glikson, Stefan Nastic, and Schahram Dustdar. 2017. Deviceless edge computing: extending serverless computing to the edge of the network. In Int. Systems and Storage Conference.
[19] Adam Hall and Umakishore Ramachandran. 2019. An execution model for serverless functions at the edge. In Int. Conf. on Internet of Things Design and Implementation. 225–236.
[20] Joseph M Hellerstein, Jose Faleiro, Joseph E Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2018. Serverless computing: One step forward, two steps back. arXiv preprint arXiv:1812.03651 (2018).
[21] Luis Felipe Herrera-Quintero, Julian Camilo Vega-Alfonso, Klaus Bodo Albert Banse, and Eduardo Carrillo Zambrano. 2018. Smart its sensor for the transportation planning based on iot approaches using serverless and microservices architecture. IEEE Intelligent Transportation Systems Magazine 10, 2 (2018), 17–27.
[22] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, and Neeraja Yadwadkar. 2019. Cloud programming simplified: A berkeley view on serverless computing. arXiv preprint arXiv:1902.03383 (2019).
[23] Young Ki Kim, M Reza HoseinyFarahabady, Young Choon Lee, and Albert Y Zomaya. 2020. Automated Fine-Grained CPU Cap Control in Serverless Computing Platform. IEEE Transactions on Parallel and Distributed Systems 31, 10 (2020), 2289–2301.
[24] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An open framework for architecting trusted execution environments. In European Conference on Computer Systems. 1–16.
[25] Philipp Leitner, Erik Wittern, Josef Spillner, and Waldemar Hummer. 2019. A mixed-method empirical study of Function-as-a-Service software development in industrial practice. Journal of Systems and Software 149 (2019), 340–359.
[26] Valentina Lenarduzzi, Jeremy Daly, Antonio Martini, Sebastiano Panichella, and Damian Andrew Tamburri. 2021. Toward a Technical Debt Conceptualization for Serverless Computing. IEEE Software 82, 1 (2021), 0–0. https://doi.org/10.1109/MS.2020.3030786
[27] Valentina Lenarduzzi and Annibale Panichella. 2021. Serverless Testing: Tool Vendors and Experts Point of View. IEEE Software 82, 1 (2021). https://doi.org/10.1109/MS.2020.3030803
[28] C Lin and H Khazaei. 2021. Modeling and Optimization of Performance and Cost of Serverless Applications. IEEE Transactions on Parallel and Distributed Systems

32, 3 (mar 2021), 615–632. https://doi.org/10.1109/TPDS.2020.3028841
[29] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. 2020. Application Management in Fog Computing Environments: A Taxonomy, Review and Future Directions. ACM Comput. Surv. 53, 4 (jul 2020). https://doi.org/10.1145/3403955
[30] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. 2017. My VM is Lighter (and Safer) than your Container. In Symp. on Operating Systems Principles.
[31] Garrett McGrath and Paul R Brenner. 2017. Serverless computing: Design, implementation, and performance. In 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW). IEEE, 405–410.
[32] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jorg Ott. 2018. Consolidate IoT edge computing with lightweight virtualization. IEEE Network 32, 1 (2018), 102–111.
[33] Stefan Nastic, Thomas Rausch, Ognjen Scekic, Schahram Dustdar, Marjan Gusev, Bojana Koteska, Magdalena Kostoska, Boro Jakimovski, Sasko Ristov, and Radu Prodan. 2017. A serverless real-time data analytics platform for edge computing. IEEE Internet Computing 21, 4 (2017), 64–71.
[34] Hai Duc Nguyen, Chaojie Zhang, Zhujun Xiao, and Andrew A Chien. 2019. Realtime Serverless: Cloud Resource Management for Bursty, Real-time Workloads. (2019).
[35] Jussi Nupponen and Davide Taibi. 2020. Serverless: What it is, what to do and what not to do. In Int. Conf. on Software Architecture Companion (ICSA-C). IEEE.
[36] Andrei Palade, Aqeel Kazmi, and Siobhán Clarke. 2019. An evaluation of open source serverless computing frameworks support at the edge. In 2019 IEEE World Congress on Services (SERVICES), Vol. 2642. IEEE, 206–211.
[37] Duarte Pinto, João Pedro Dias, and Hugo Sereno Ferreira. 2018. Dynamic allocation of serverless functions in IoT environments. In 2018 IEEE 16th International Conference on Embedded and Ubiquitous Computing (EUC). IEEE, 1–8.
[38] R Arokia Paul Rajan. 2020. A review on serverless architectures-function as a service (FaaS) in cloud computing. TELKOMNIKA 18, 1 (2020), 530–537.
[39] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. 2019. Serverless Computing: A Survey of Opportunities, Challenges and Applications. arXiv:1911.01296 [cs.NI]
[40] Davide Taibi, Nabil El Ioini, Claus Pahl, and Jan Raphael Schmid Niederkofler. 2020. Patterns for Serverless Functions (Function-as-a-Service): A Multivocal Literature Review. In Int. Conf. on Cloud Computing and Services Science (CLOSER 2020). 181–192.
[41] Davide Taibi, Nabil El Ioini, Claus Pahl, and Jan Raphael Schmid Niederkofler. 2020. Serverless Cloud Computing (Function-as-a-Service) Patterns: A Multivocal Literature Review. In Int. Conf. on Cloud Computing and Services Science (CLOSER'20).
[42] Davide Taibi, J. Spillner, and K. Wawruch. 2021. Serverless Where are we now and where are we heading? IEEE Software 38, 1 (2021).
[43] Erwin Van Eyk, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Ut,
ă, and Alexandru Iosup. 2018. Serverless is more: From paas to present cloud computing. IEEE Internet Computing 22, 5 (2018), 8–17.

[44] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P Jue. 2019. All one needs to know about fog computing and related edge computing paradigms: A complete survey. Journal of Systems Architecture 98 (2019), 289–330.

45. Hassan, N., Gillani, S., Ahmed, E., Yaqoob, I., Imran, M.: The role of edge computing in internet of things. IEEE Communications Magazine 56(11), 110{115 (November 2018). https://doi.org/10.1109/MCOM.2018.1700906
46. H□astbacka, D., Halme, J., Larranaga, M., More, R., Mesi□a, H., Bj□orkbom, M., Barna, L., Pettinen, H., Elo, M., Jaatinen, A., Hoikka, H.: Dynamic and exible data acquisition and data analytics system software architecture. In: 2019 IEEE SENSORS. pp. 1{4 (Oct 2019). https://doi.org/10.1109/SENSORS43011.2019.8956662
47. Liu, M., Yu, F.R., Teng, Y., Leung, V.C.M., Song, M.: Distributed resource allocation in blockchain-based video streaming systems with mobile edge computing. IEEE Transactions on Wireless Communications 18(1), 695{708 (Jan 2019). https://doi.org/10.1109/TWC.2018.2885266
48. Nastic, S., Rausch, T., Scekic, O., Dustdar, S., Gusev, M., Koteska, B., Kostoska, M., Jakimovski, B., Ristov, S., Prodan, R.: A serverless real-time data analytics platform for edge computing. IEEE Internet Computing 21(4), 64{71 (2017). https://doi.org/10.1109/MIC.2017.2911430
49. Ning, H., Li, Y., Shi, F., Yang, L.T.: Heterogeneous edge computing open platforms and tools for internet of things. Future Generation Computer Systems 106, 67 { 76 (2020). https://doi.org/https://doi.org/10.1016/j.future.2019.12.036

50. Nupponen, J., Taibi, D.: Serverless: What it is, what to do and what not to do. In: IEEE International Conference on Software Architecture (ICSA 2020) (03 2020)
51. Palade, A., Kazmi, A., Clarke, S.: An evaluation of open source serverless computing frameworks support at the edge. In: 2019 IEEE World Congress on Services (SERVICES). vol. 2642-939X, pp. 206{211 (July 2019). https://doi.org/10.1109/SERVICES.2019.00057
52. Taibi, D., El Ioini, N., Pahl, C., Schmid Niederkler, J.R.: Serverless cloud computing (function-as-a-service) patterns: A multivocal literature review. International Conference on Cloud Computing and Services Science (CLOSER2020) (2020)
53. White, G., Cabrera, C., Palade, A., Clarke, S.: Augmented reality in iot. In: Liu, X., Mrissa, M., Zhang, L., Benslimane, D., Ghose, A., Wang, Z., Bucchiarone, A., Zhang, W., Zou, Y., Yu, Q. (eds.) Service-Oriented Computing { ICSOC 2018 Workshops. pp. 149{160. Springer International Publishing, Cham (2019)

54. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., Suter, P.: Serverless computing: Current trends and open problems. In: Research Advances in Cloud Computing (2017)
55 Baresi, L., Mendon_ca, D.F.: Towards a serverless platform for edge computing. In: 2019 IEEE International Conference on Fog Computing (ICFC). pp. 1{10. IEEE (2019)
56. Cheng, B., Fuerst, J., Solmaz, G., Sanada, T.: Fog function: Serverless fog computing for data intensive iot services. In: 2019 IEEE International Conference on

Services Computing (SCC). pp. 28{35. IEEE (2019)
57. Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., Weihl,B.: Globally distributed content delivery. IEEE Internet Computing
6(5), 50{58 (Sep 2002). https://doi.org/10.1109/MIC.2002.1036038,
https://doi.org/10.1109/MIC.2002.1036038