International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.56472/WCAI25-144
Eureka Vision Publication | WCAI25-Conference Proceeding

Original Article

Multi-Cloud Serverless Computing & FaaS Architectures for Resilient and Cost-Efficient Systems

Mr. Anil Kumar Manukonda
IT professional from USA.

Abstract - The multi-cloud serverless computing involves the placement of Function-as-a-Service (FaaS) workloads with different cloud providers to increase system resilience and a cost-optimized environment. This paper will give a detailed summary of multi-cloud serverless architecture, its motivations, design, and challenges as well as its future. We explain how several clouds can help eliminate the risks of vendor lock-in and provider outages (which is a top issue in such high-availability and compliance-driven industries as fintech). The Introduction includes the description of the relevance of multi-cloud FaaS and the current specific drivers, including the resilience of operations and cost reduction. Literature Review investigates the research available about the serverless methodology and multi-cloud access strategies further demonstrating the tradeoffs regarding performance and economics. An Architectural Overview explains the design of multi-cloud FaaS platform architectures and the aspect of orchestration layers that helps achieve interoperability between AWS Lambda, Azure Functions, Google Cloud Functions and others. We explore some of the Key Challenges such as cold starts, performance overhead, observability, security, and regulatory compliance of multi-cloud deployment. Cost Optimization Strategies are discussed with reference to the pricing strategies at major providers and the methods to cut the cost with the maximum fault tolerance. Resilience and Fault-Tolerance patterns are then discussed including active-active deployments and distributed failover across clouds. Case Studies on fintech present another demonstration of how multi-cloud serverless enhances compliance and reliability in payment systems. Last but not least, we can point out where Future Directions developing in the form of edge computing and AI-based optimizations or open-source FaaS frameworks will influence the next wave of resilient and cost-effective multi cloud serverless platforms.

Keywords - Multi-Cloud Computing, Serverless Computing, Function-as-a-Service (FaaS), Virtual Serverless Provider (VSP), AWS Lambda, Azure Functions, Google Cloud Functions, Cloud Orchestration, Event Bridge, Scheduler, CloudEvents, Cost Optimization, Free Tier Utilization, Pricing Arbitrage, Cloud Portability, Vendor Lock-In, Cold Start Latency, Performance Overhead, Observability, OpenTelemetry, Debugging, Identity Federation, Security, Compliance, GDPR, Digital Operational Resilience Act (DORA), Data Residency, Fault Tolerance, Active-Active Deployment, Disaster Recovery, DNS Failover, Global Load Balancing, Service Mesh, Edge Computing, AI-based Optimization, WebAssembly (WASM), Knative, OpenWhisk, Kubernetes, Infrastructure as Code, Terraform, Serverless Framework, Monitoring, Logging, Elastic Stack, Datadog, Open Policy Agent (OPA), CloudEvents Standard, FinTech Applications, Payment Systems, Trading Platforms, Open Banking, Fraud Detection, Cost-Efficient Architectures, Scalability, Reliability, Governance, Future Trends.

1. Introduction

Multi-cloud serverless computing refers to the act of having serverless applications (Faas and associated services) that operate across multiple cloud providers at the same time. In serverless, users express their application in discrete functions using one of the managed platforms (e.g., AWS Lambda, Azure Functions, Google Cloud Functions) without provisioning and operating servers. This paradigm is extended in multi-cloud deployments in which the same codebase or workload can be deployed to various clouds with only slight adjustments [11]. In this, infrastructure management is abstracted and provides stronger portability, such as, a Python function can be pushed to both AWS and Azure with just configuration and not logic change [11]. The possibility of executing the same functions on more than one provider is a great opportunity of resilience and cost efficiency.

Multi-cloud serverless is implemented in an organization based on a number of reasons. The first one is to prevent vendor lock-ins, which is an intrinsic problem of serverless because of the proprietary FaaS and Backend-as-a-Service (BaaS) integrations. Even with single-cloud serverless workflows the BaaS in a given cloud (e.g. storage, databases) can only generally call functions on that cloud, and therefore cannot feasibly port the application to another provider. Multi-cloud policies mitigate this analysis by distributing the loads among different providers, lessening the reliance on a particular provider ecosystem. The other motivation is operational resilience. Multi-cloud functionality can be used to run functions in more than one cloud to increase availability - should one cloud provider be experiencing an outage or decline in performance, functions may be failed over to another cloud provider [2]. This is particularly vital in such industries as financial service where any downtime converts to lose revenue as well as scrutiny by regulators. In fact, banks and other financial institutions are simply augmenting cloud-parallel serverless systems on other cloud platforms with the clear purpose of circumventing the concentration risk and preserving their operations in the event of failure [7][8].

Servers that are using multi-cloud serverless also allow to optimize costs [1]. The performance features of cloud providers and pricing are different. Organizations can take advantage of price gaps and promotions using the dynamic provider selection schemes that would maintain the optimum provider per specific criterion or workload. Virtual Serverless Providers (VSPs) Research prototypes of FaaS services across clouds have demonstrated that cost-performance can be enhanced by aggregating offerings, e.g. increasing throughput by up to 4× and decreasing the cost of invocations by >50% by sending to those providers that are most efficient at each time [2]. Also, the free tier (usually millions of free invocations per month) of any cloud may be used in any combination, which may decrease the expenses on occasional workloads. Diversity of services and regions provided by each of the providers can be accessed beyond the cost and reliability variables offered by multi-cloud deployments. One cloud may provide more accelerated GPUs or even particular AI services whereas the other one has data centers in a needed jurisdiction; the combination of these two clouds can lead to a better solution. To sum it up, a multi-cloud serverless approach offers us a kind of best of all worlds: high availability, cost-effectiveness, no lock-ins and flexibility to fit various regulatory and performance requirements [1][7].

Simultaneously, multi-cloud serverless computing is relatively complex. This paper discusses these issues in more detail and the state-of-the-art solutions to the problems in the subsequent sections. We start with the background and related work on serverless systems and multi-cloud approach. We subsequently give a high-level design of multi-cloud FaaS platforms and discuss important technical challenges of multi-cloud FaaS platforms, including cold start latency, cross-cloud monitoring, security, and data management. Multi-cloud FaaS and cost optimization and fault tolerance strategies are discussed in detail, and viewed through the prism of practical use in a fintech-related scenario. As our final conclusion, we address the emerging trends, such as integrating edge computing, AI-based scheduling, and the use of open-source framework strategies, which lead to the adoption of future resilient and cost-efficient multi-cloud serverless systems.

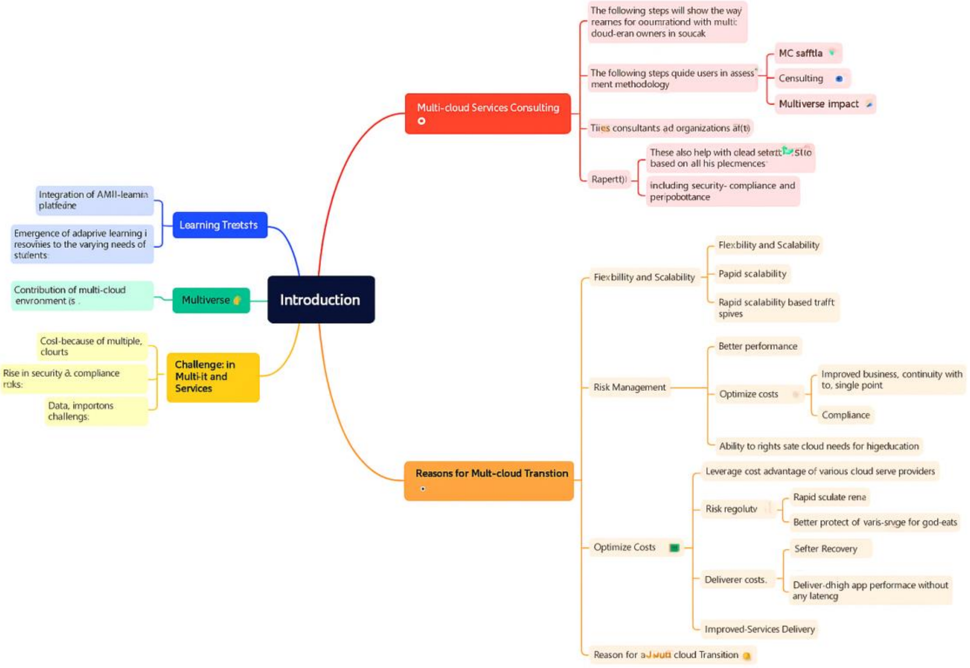

Introduction
Multi-cloud Services Consulting
The following steps will show the way reames for oournrationd with multi: doud-eran owners in soucak
The following steps quide users in assess ment methodology
MC safftla
Censulting
Multiverse impact
Tiies consultants ad organizations af(t)
Rapertt)
These also help with clead setett Stto based on all his plecmences
including security- compliance and peripobottance
Learning Treatsts
Integration of AMII-leamin platfedne
Emergence of adaprive learning i resovnies to the varying needs of students
Multiverse
Contribution of multi-cloud envronment (s .
Challenge: in Multi-it and Services
Cosl-because of multiple, clourts
Rise in security & compliance roks:
Data, importons challengs:
Reasons for Mult-cloud Transtion
Fiexbillity and Scalability
Flexbility and Scalability
Papid scalability
Rapid scalability based trafft spives
Risk Management
Better performance
Optimize costs
Improved business, continuity with to, single point
Compliance
Ability to rights sate cloud needs for higeducation
Optimize Costs
Leverage cost advantage of various cloud serve providers
Risk regolutv
Rapid sculate rena
Better protect of varis-srvge for god-eats
Deliverer costs.
Sefter Recovery
Deliver-dhigh app performace without any latencg
Improved-Services Delivery
Reason for aJ wuti cloud Transition

Figure 1. Mind Map of Multi-Cloud Serverless Computing

2. Architectural Overview

On a high level, a multicloud serverless framework brings an abstraction on top of FaaS functionality of single cloud providers. This orchestration (or aggregation) layer is intended to offer a single execution environment to developers and to coordinate the deployment of activities and invocation of functions in many clouds. Figure 1 suggests a conceptual architecture of such a platform commonly known as a Virtual Serverless Provider (VSP). The orchestration level includes a number of important elements [2]:

- **Scheduler/Router:** A focal part that determines where to run every invocation of functions. The scheduler can analyze and run on real-time performance measurements and cost information to pick the best cloud provider to accept an incoming request. The policies may be latency minimization, cost minimization or balancing the load between the providers to have redundancy [2].
- **Event Bridge:** With a multi-cloud workflow, events and trigger must be directed to the right cloud. Event bridge component normalizes event format (such as CNCF CloudEvents format) and relays events (originating in some cloud) to functions in another cloud when required. As an example, an AWS SNS topic message could send an Azure Function via this bridge.
- **Controller & Deployment Manager:** The task of this sub-layer is to coordinate the lifecycle of functions across clouds: to deploy the code to each provider, to update the configurations, to guarantee availability of necessary run-time environments on all targeted platforms. The deployment tools such as serverless Framework or Terraform are frequently applied to this purpose and define the functions in the provider-agnostic way [11].
- **Performance Monitor:** In the decisions being made on the basis of provider performance, the system measures, on a regular basis, the time of execution of functions, the length of queues, rate of errors etc, per each cloud. This monitor provides data (ex: average latency on AWS vs AWZ on the past 5 min) to the scheduler to allow adaptive routing [2].
- **Global Cache and State Management:** The frequently accessed data or even the results of any functions may be stored on a cache layer so as to prevent cross-cloud calls. Since reading data to a dis-located cloud introduces latency and egress charges, a distributed caching approach (potentially using a cloud-impartial datastore such as Redis or MongoDB Atlas) avoids bring the data near the execution of functions. Other multi-cloud serverless systems merely externalize all state to a third party storage (externalized to any cloud storage), such as a neutral storage service (Cloudflare R2 or Cassandra) that functions on different clouds can share data without continuously copying through cloud specific storage [11].
- **Billing and Usage Aggregation:** Usage in each of the clouds will be charged individually. VSP layer can conceal this by totalizing expenses and maybe pay the consumer one total amount of bill. The other metric it records is by provider usage, which can be useful as a means to optimize free tier usage on each.
- **Authentication & Identity Federation:** The integrated identity, as well as access management system is required according to which the invocation of functions and calls among the services is possible within clouds in a secure way. This includes dealing with API tokens/credentials with each provider and potentially issuing identity tokens that will be trusted across more than one cloud in a workflow.

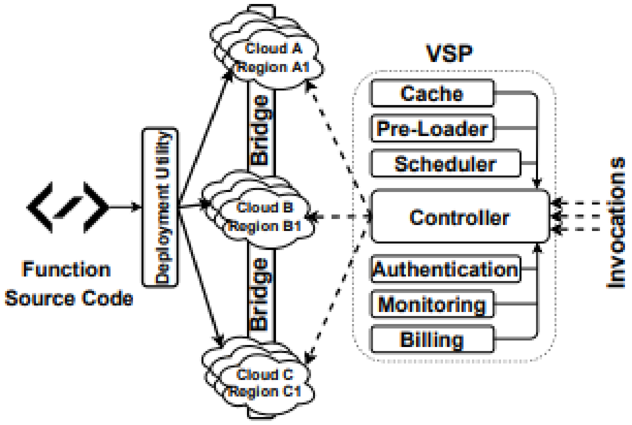


Figure 2. The high-level overview of the proposed VSP.

Practically, multi-cloud serverless architecture implies two primary approaches to it:

- **Federated Functions, Central Orchestrator:** Here, the copies of all functions and possibly the representation services are executed on respective cloud(s) and externally (or client-side) logic responds to requests by re-directing them to a particular cloud. A real world example involves running an HTTP API as a Lambda and an Azure Function; a worldwide API gateway or DNS load balances the traffic between the two depending on whether one of the functions is unavailable or via geography. The orchestrator (e.g. as a layer in the client or a custom proxy), guarantees that only a single provider is actually working on a particular request, yet many are hot and ready. This is similar to application-level active-active fail over. The technology of Netflix [Werner?] - multiregional (multi-cloud) traffic routing using a short-time failsafe is an applicable analogy.

- **Unified Platform (Third-Party Broker):** In this case there is one logical platform across clouds, the developer deploys its logic to the meta-platform, and it in the background, deploys to many clouds and invokes where necessary. Such a model is the VSP [2]. A layer of cloud-agnostic can be built using technologies such as Knative (that can be run on any Kubernetes cluster) or OpenWhisk. To cite an example, an open-source FaaS gateway may accept an event and call a Lambda or a Google Cloud Function through API calls. The user has contact to the gateway only with the API, but not the API of every cloud.

With this type of architecture, it is difficult to realize cloud vendor interoperability. Every cloud provider is based on different event sources, different signatures of functions, resource constraints, and monitoring tools. The problem of portability occurs on many levels:

- **Function Code Portability:** Very basic (stateless Python/Node scripts being one example) applications can be ported with no issue, but anything that has an AWS-specific SDKS or other service dependency (call to AWS DynamoDB, for example) is not portable. The best practices promote the usage of abstract interfaces or external services. As an example, one may refer to a storage API (cloud-neutral) instead of AWS S3 itself.
- **Configuration & Deployment:** various providers need different configuration (size of memory, timeout, environment variables). This is reduced by the use of tools such as the Serverless Framework where a single configuration file can be used to directly target multiple clouds with sections per provider where applicable [11]. Not everything is one-to-one also, such as one cloud may have 15-minute max execution time (AWS) and the other only 5 minutes (existing Azure Functions) so that workflows may also need to be redesigned in order to be equivalent.
- **Networking & Integration:** In case spans clouds, communication may be across a public internet or special interconnected used. Other businesses embrace a service mesh overlay, which provides a connection between functions within the environments. Recent architecture In a recent proposal named Serverless Mesh, serverless functions across clouds would be tied together with a common service mesh layer to provide dynamic routing and policy application between environments. This is capable of processing function-to-function calls securely and efficiently, but in the same way that service mesh operates microservices within a cluster.

To conclude, multi-cloud serverless architecture enables the concept of serverless computing into the multi-provider context by providing an abstraction level through orchestration. And regardless of how it has been achieved (through third-party services, open-source frameworks, or simple trickery involving CI/CD pipelines and routing), the net objective is to reduce multiple clouds to a single logical FaaS environment. In the following, we will look at the real technical issues of such architectures.

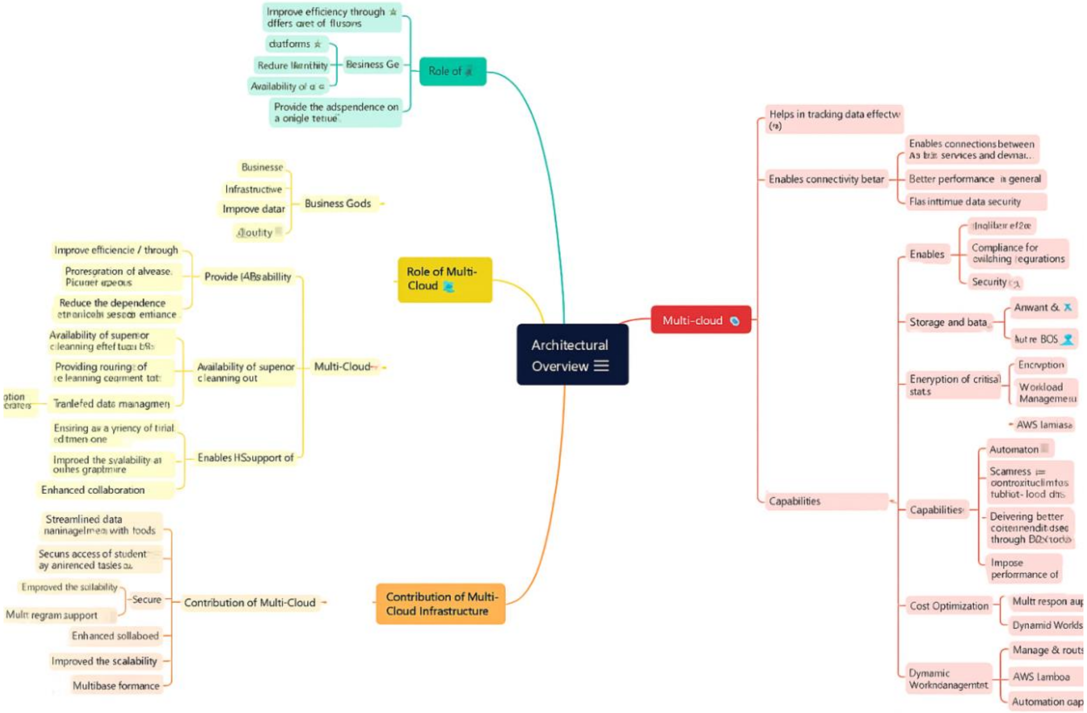

Improve efficiency through dffers caret of flusons
dutforms
Redure lilentity
Besiness Ge
Availability of a
Provide the adspendence on a onigle tetiue
Role of
Businesse
Infrastructwe
Improve datar
Business Gods
Aloutity
Improve efficiencie / through
Proresgration of alvease. Picuaer epeous
Provide IABs abillity
Reduce the dependence etnanicobi sesocb entiance
Role of Multi-Cloud
Availability of superior cleanning eftef tuoi b
Providing rouring of re leanning ceament tot
Availability of supenor cleanning out
Multi-Cloud
otion erators
Tranlefed data managmen
Ensiring av a yriency of tirial edtmen one
Improed the svalability ai oihes graptmire
Enables HSsupport of
Enhanced collaboration
Streamlined data naninagelmen with toods
Secuns access of student ay anirenced tasles
Emproved the suilability
Secure
Contribution of Multi-Cloud
Multr regram support
Contribution of Multi-Cloud Infrastructure
Enhanced sollaboed
Improved the scalability
Multibase formance
Architectural Overview
Multi-cloud
Helps in tracking data effectw
Enables connections between As bis services and devnat..
Enables connectivity betar
Better performance in general
Flas inttimue data security
Inglibeir ef2ce
Enables
Compliance for ovilching requrations
Security
Anwant &
Storage and bata
Aut re BOS
Encrvption
Eneryption of critisal stats
Workload Managemen
AWS lamiasa
Automaton
Scamress controxituclimtos tubliot- lood dns
Capabilities
Capabilities
Deivering better coitennendit dsec through DDX toda
Impose performance of
Cost Optimization
Multr respon aup
Dynamid Worlds
Manage & routs
Dymamic Workndanagemtet
AWS Lamboa
Automation oap

Figure 3. Mind Map of Architectural Overview for Multi-Cloud Serverless Computing

2.1. Key Challenges

Although multi-cloud FaaS systems are potent, these solutions are exposed to various technical difficulties that should be overcome before the full potentials can be met:

2.1.1. Vendor Lock-In and Portability

- **Vendor Lock-In:** Serverless services by nature, bind the user more firmly to the ecosystem of the specific provider than higher services. The cloud manages exeution, to the extent that it requires definite event, data store, and function packaging interfaces as well. As mentioned, a BaaS service (such as an object store or database) is generally directly able to initiate functions in the same cloud only. Therefore, an application composed of, e.g. AWS API Gateway -> AWS Lambda -> AWS DynamoDB is already locked-in both at the code and the event wiring levels. A simple solution to this lock-in is multi-cloud deployment: by keeping the application in multiple clouds and not moving the whole application, one would still have the freedom to change in case it is necessary. Yet it is hard to make actually cloud-agnostic functions. The vanilla thing of IAM (Identity and Access Management) is different among clouds; transferring a capability might run into a dead-end in case an appropriate service or permissioning model does not exist on the destination cloud. Yussupov et al. (2019) observed numerous portability hazards where even insignificant serverless applications enclosed broken dependencies with serverless transfer between providers [1]. To allay this, developers are advised to limit themselves to common denominators (such as use of HTTP-based triggers over proprietary event buses) and use frameworks which abstract provider details. The portability can also be enhanced by containerization of functions (e.g. by using Docker with AWS Lambda container-support or Google Cloud Run) but this comes with an operational overhead again.
- **Workload Portability:** It is not trivial to migrate a live workload between clouds without downtime even with portable code. Data must be mirrored or at least available in the two environments and that stateful connections (e.g. WebSockets or long-running streams) cannot be easily re-directed between clouds. Other designs take an Active-Passive position: maintain one cloud as primary, and only spin up the capability onto a different cloud in the event of fail over. This restraints lock-in without necessarily exploiting the full potential of multi-cloud. This can more dynamically be done by also running active instances in all clouds but using a clever routing layer (e.g. a global load balancer or DNS routing with health checks) to direct traffic. It results in consistency (without making sure that all the instances will run with the same version of code/config) and idempotency (as the concurrent clouds-run execution will cause unintended duplication of an event processing). The latter are possible to deal with relying on at-least-once semantics and deduplication mechanisms, or via functions that are designed to handle multiple occurrence of events.

2.1.2. Cold Starts and Performance Overhead

- **Cold Starts:** One infamous problem with serverless computing is the cold start latency, the time it takes to initalise a function container after having gone idle. Cold starts might be intensified by multi-cloud deployment in two ways. To start with, distributing the traffic to a couple of providers may lead to the invocation of each less frequently, which implies functions will spend a longer time in an idle state, sufficient to be displaced and cause cold start to occur when traffic is suddenly high. Second, during failover of cloud to cloud (where Cloud A crashes and Cloud B is now supposed to pick up all traffic), functions on Cloud B will be cold (not been used before) and experience startup times. Each provider has a set of cold start properties: experiments have found that AWS cold starting languages such as Node.js occur within a few hundred milliseconds on average, but both Azure and Google Cloud Platform can cold start up to a second or more. Methods of reducing cold starts involve avoiding them altogether (keeping functions warm, or invoking them frequently but only periodically, e.g. through ping invocation) or having them provisioned (with a provider reserving ideally-tempered instances at a higher cost). At least in the multi-cloud scenario, it may be possible to keep at least one instance warmed on every provider (at the price of higher base cost, but at least there is a faster failover). Certain studies indicate that it is possible to pre-smartly load functions on other clouds in the event of a slowdown in main cloud [2]. The trade-off between cold-starts overhead and cost in practice is an art, especially on highly latency-sensitive fintech services where even a non-zero amount of always-warm redundancy between clouds is worth paying to eliminate any delay users would experience.
- **Performance Overhead:** The inescapable introduction of latency due to inserting an orchestration layer and cross-cloud communication causes a delay as compared to a single-cloud that makes calls. To give an example, when an Azure Function requires accessing data in an AWS S3 bucket the network request crosses the internet (or costly interconnects) and this will be slower than actually locating the data in an Azure Blob Storage. The multi-cloud deployment could have additional hops: an event enters the queue of Cloud A, calling the function of Cloud B through the event broker - the latency of each such transition is several milliseconds. Another element in the request path that may become a bottleneck is the orchestration logic (in case used as a proxy or a gateway). Such overheads should consider being reduced to the lowest possible by implementing the orchestration layer in a most optimal way (e.g., using it in edge points near clients, or in every cloud to leverage its internal high-speed network). Service mesh techniques are interesting in this case, since they

can re-route calls reliably to local or remote functions and implement circuit breaking policies in case of a slow cloud. Resources are also heterogeneous (CPU speed, memory performance, etc.) in a given cloud and even a region [1]. One may have an application that can run slower on one providers infrastructure than another, and the scheduler should not only consider a momentary load but natural differences in speed. Uncontrolled multi-cloud may also result in fluctuating response time in case of certain occasions, a slower platform is taken to reduce cost.

2.1.3. Observability and Debugging

- **Observability:** Serverless application monitoring is already a problematic topic within a single cloud because of the distributed desire to surveil a network and the ephemeral character of serverless applications. Observability becomes worse still in a multi-cloud environment with different providers having their own monitoring and no in-built way of correlating events between them (CloudWatch and AWS, Azure Monitor, and Google Cloud Logging, and so on). This will result in a system segmented perspective. Example, a transaction can be accomplished through an AWS function calling a GCP one, then in case of a failure, it goes through your tracking journey and logs in these two systems, which is an unnecessary hassle. It is important to centralize logs and metrics. The solutions are exporting all logs to a single external system (such as Elastic Stack or Datadog) or relying on observability platforms that are cloud-agnostic, taking in data across clouds. New standards such as OpenTelemetry offer vendor-neutral tracing; with it, it is possible to propagate a trace context across functions even across clouds, and then reassemble the end-to-end trace in an outside analysis tool. This, however, presupposes that functions are instrumented with the correct SDKs and that there is a trace collector in every environment.
- **Debugging:** There are special challenges to troubleshooting multi-cloud serverless applications. One may need to set up test events in several clouds and handle a variety of debugging tools to reproduce an error. Besides, when a problem occurs, it may not be necessarily clear whose end is this in the function code, or in the orchestration on the way between the functions, or it is a cloud-related peculiarity. Depending on the environment of each individual provider (runtime versions, memory allocation strategies) behavioral differences might arise. As an example, a Python function may end up having a memory leak problem on one platform (because of the way the runtime is handled). The following techniques can alleviate debugging: deployment of a common test harness that can call functions in all clouds with the same inputs and compare outputs, feature flags that can be used to turn providers on or off and continuous testing across all environments. Part of the multi-cloud frameworks tries to give a unified CLI or dashboard to configure and control all the functionality on the clouds, which can assist in debugging, as it gives a one-point view. However, it is a tall order that engineers will mostly be required to have proficiency in several cloud systems to be good debuggers.

2.1.4. Security and Compliance

- **Security:** The transition to use of many clouds increases the number of attack points and makes security management difficult. All the clouds of the world possess their security models - the distinctions in IAM positions, permission scopes, secret administration, and network insurance (VPCs, subnets) imply that a multi-cloud application should abide by all of them at the same time. The problem of least privilege access in functions across clouds is somewhat hard; the roles/policies to achieve this need to be kept in each cloud, and it is easy to configure this incorrectly. The secrets (API keys, database credentials) may also need to be replicated to the secret stores of several clouds and, therefore, should be synchronized. An important suggestion is to get an external secrets manager or vault against which all functions (regardless of where they may run) can safely access secrets. One more danger is the possibility of denial-of-wallet attacks that is even greater in multi-cloud: the attacker launches functions in all your clouds and, therefore, your expenses may be multiplied (this has similarities with the denial-of-service, but which acts against the pay-per-use model). Such attacks require uniform throttling and anomaly detection over cloud providers - e.g. with standardized defensive measures or with a layer of third-party security (that is insensitive to who the provider is) that looks at whether particular endpoints within its network or cloud are experiencing unusably high traffic independently of the provider.
- **Compliance:** In cross region or global deployments, there are restrictions on regulatory compliance. Also, a multi-cloud strategy is inherently global by intention, and in some cases, one driver is an interest in using a cloud region in a particular country in order to meet a localization requirement. Nonetheless, the issue of compliance agility among companies implies that each cloud is configured according to the standards (ISO, SOC2, GDPR, etc.). The Digital Operational Resilience Act (DORA) adopted by EU explicitly mentions the need to manage the third-party risk (where cloud concentration risk also falls) [8]. That can be resolved by not putting all their eggs into one basket using multi-cloud, but then the regulators will want to know whether your failover actually works and whether data is equally safe at all locations. However, consistent compliance may entail encryption of all data in transit and at rest across all clouds (therefore even in case one cloud does not meet a standard, it is covered by your own encryption). Also, the regulations on cross-border data transfer (such as GDPR) may be problematic in case of an unintentional processing of data of an EU user on a German cloud by a U.S. cloud in case of a failover. Such careful design is required so that some of the data can

be anchored to some regions or providers. Banks with multi-cloud adopt compute across clouds but not between sensitive data on the cloud platforms e.g. a database replicated in the compliant zone of each cloud (not intermingled across the two clouds), and will only create state within the compliant zone and will shift clouds only at compute level when and as required. Auditing also must be done in a compliant manner: acts done on each cloud must be recorded and auditable in the central framework in order to meet oversight needs.

3. Network and Latency

There are also several additional difficulties that should be mentioned, including the use of networking in multi-cloud deployment which can be quite complex since it might be necessary to connect functions across cloud boundaries, which would either imply leveraging the internet or establishing private connectivity (e.g., someone would have AWS Direct Connect to an Equinix that also connects to Azure Express Route). When it comes to actual working, network latency across clouds may form a bottleneck; a multi-cloud chain of functions may not facilitate high frequency trading across continents, eg. because of latency. This can be alleviated by solutions such as locating functions as near as possible to where data has been located (data gravity considerations) or to using content delivery networks or edge servers to route requests in an optimal way.

A more complex configuration may involve Anycast DNS or worldwide load balancers (such as Cloudflare) that will be able to send the request of the user to the closest healthy deployment of the cloud, to reduce the number of cross-cloud hops. Building multi-cloud serverless systems presents many challenges: cold start, compliance, and many others that need to be planned carefully and, in some cases, are customized to meet the specific needs. In our following chapter, we see how cost optimization is still possible despite some of such obstacles, and multi-cloud environments may be cost effective or least, have better value than the price.

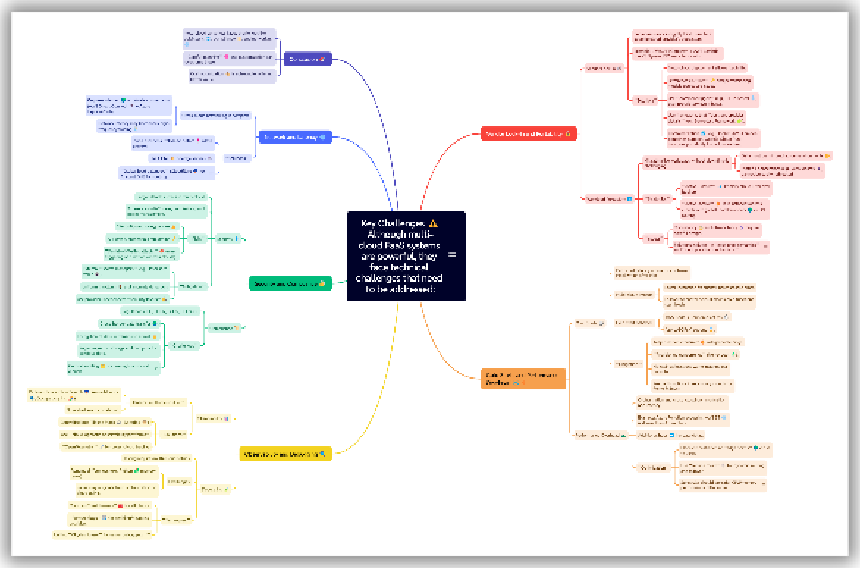

Key Challenges ⚠ Although multi-cloud FaaS systems are powerful, they face technical challenges that need to be addressed:

Figure 4. Mind Map of Key Challenges in Multi-Cloud Serverless Computing

3.1. Cost Optimization Strategies

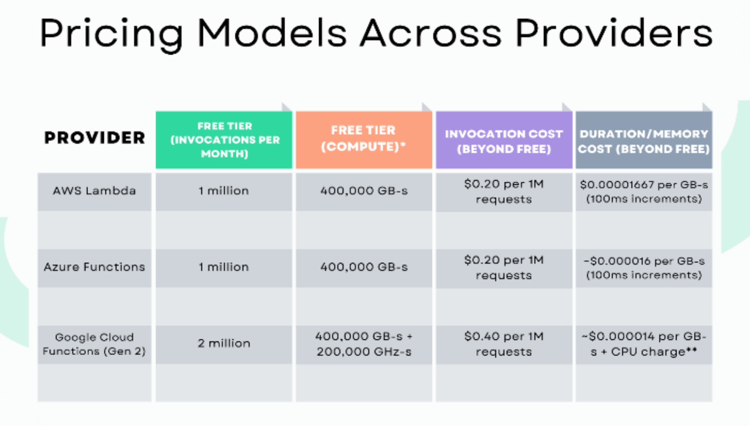
Another of the attractions of serverless is the cost model: you only pay what you actually use, in great detail. Multiple-cloud deployment gives more tuning handles of cost optimization, though savings can only be achieved after knowing the differences in prices of various providers and workload placement.

3.1.1. Pricing Models across Providers

The three key clouds FaaS is available in at the time of writing for pricing (2025) is shown in Table 1. Each of them provides a free tier and charges on an invocations-and-resource-time basis (memory/CPU):

Table 1 indicates that prices are almost the same in AWS and Azure though not accidentally since competition has created similar rates. The free level from Google doubles the amount of free invocations, and this could be useful to workloads having large amounts of short-duration invocations. Still, the post-free rates of Google have a CPU component (up to a degree billing

CPU at ~$0.000010 per GHz-s) that complicates direct comparison. Insightful piece of information: Cost per execution may differ among providers due to the size of memory, and the time of execution. To illustrate, a workload with intensive CPU may be more expensive on GCP as a result of GHz-second pricing approach, when on AWS/Azure such a workload could be nothing more than a memory-duration pay.



Pricing Models Across Providers

PROVIDER	FREE TIER (INVOCATIONS PER MONTH)	FREE TIER (COMPUTE)*	INVOCATION COST (BEYOND FREE)	DURATION/MEMORY COST (BEYOND FREE)
AWS Lambda	1 million	400,000 GB-s	$0.20 per 1M requests	$0.00001667 per GB-s (100ms increments)
Azure Functions	1 million	400,000 GB-s	$0.20 per 1M requests	~$0.000016 per GB-s (100ms increments)
Google Cloud Functions (Gen 2)	2 million	400,000 GB-s + 200,000 GHz-s	$0.40 per 1M requests	~$0.000014 per GB-s + CPU charge**

Figure 5. Pricing Models across Providers [11]

3.1.2. Multi-Cloud Cost Savings

Multi-cloud strategies present cost savings by being able to save in the following ways:

- **Leveraging Free Tiers:** The free levels of each provider can be stacked. In case your application faces moderate traffic you may assign a portion of request to each of the clouds so that none of them exceed the free level. An example is an application that has 2M requests per month and can host 1M on AWS and the other 1M on Azure at no cost on either cloud (rather than $0.20 on each extra 1M requests on a single cloud). On the same note, when you partition the usage of memory-time you use the free GB-seconds in both platforms. This would necessitate a smart traffic router, which could save huge sums on low and medium traffic applications.
- **Pricing Arbitrage:** Sometimes a cloud would just be less expensive to a certain workload. As an example, a function that has high CPU, albeit for a short duration, may be less expensive on AWS (through which you can allocate memory to have more CPU at the same price per GB-s) as compared to the CPU Credits pricing on GCP. However, the extreme number of short invocations would favor GCP with larger allowance of free calls. Profiling the use of a given functionality on each of the platforms (which can be accomplished by using the End Analysis System developed by Zhao et al.), it is possible to determine the cheapest service to provide said functionality. A multi-cloud planner can then accelerate consumption of the least costly platform until it reaches some limit (e.g. second tier of a discount or settled capacity).
- **Geographic Cost Differences:** Cloud providers sometimes charge region specific or data egress prices. In case your users are all over the world, using regional functions in various clouds may save network egress charges (as transit within-region or within-provider is typically less expensive or even free). As an example, transmission of data between an AWS Lambda in Virginia and an AWS S3 in Virginia will be free whereas it would cost egress on Azure and ingress on AWS with transmission between Azure and AWS. A cost-effective design may take the request of a user to a cloud with a data center that is nearest or where data is located, and not cross transfer cloud bits except when required.
- **Transient Spot Instances vs. Serverless:** While serverless itself doesn't have a "spot market", a multi-cloud setup can mimic one by shifting load to whichever provider currently offers better cost-performance. Providers sometimes have promotions or varying performance (if one cloud's infrastructure is underutilized, your functions might run faster, effectively costing you less in GB-s). A sophisticated AI-driven cost optimizer could learn patterns – for instance, perhaps at certain hours GCP functions execute faster (less load) so you save time-based fees, whereas AWS might slow down at those times. Scheduling more tasks on GCP in that window would save cost. This is a forward-looking concept, but research suggests it's feasible to adapt scheduling to such signals [2].

3.1.3. Reserved Capacity and Discounts:

Although serverless lacks a so-called spot market, a multi-cloud environment can emulate it with the load redistribution to the provider that provides the best cost-performance at any point in time [1]. At times, there are offers offered by the providers or different performance (in case a cloud provider has one infrastructure that has less work, then your functions can go at a faster pace which will cost you less in GB-s). An advanced AI-powered cost optimizer would be able to recognize patterns, i.e. maybe in the given hours the GCP functions run faster (less load) so you save time-based costs, or maybe AWS becomes slower during the hours. It will be cost-effective to schedule additional work on GCP during the same time slot. It is definitely a visionary idea and yet there is evidence to it that it could be adapted to schedule anything to such signals.

3.1.4. Efficient Function Design

Optimizations around standard cost are also important, and even more in multi-cloud:

- **Optimize execution time:** Fewer functions created means less spending on any cloud. This includes code optimization, the use of faster runtimes (e.g., a new programming language or version of a runtime when one of the cloud runtimes is faster), memory right-sizing (i.e., providing a piece of code the amount of memory it needs to be fast but not so much that memory costs go up significantly in GB-s). Luckily, experimentation is possible in the context of multi-cloud deployment, and it is possible to test the performance of a specific function on each platform and tailor memory/CPU parameters to the test conditions.
- **Minimize data transfer:** As mentioned, data egress may cost a lot. Multi-cloud deployment systems ought to consider doing data-processing where it exists or probably take advantage of the available open data endpoints to evade-dual payments. To ensure that repeated calls to access the same data do not fail to do so over clouds, results caching can reduce the cost of transfer; so can data compression [11].
- **Auto-scheduling and autoscaling:** Any individual cloud will automatically autoscale its operations, however lower-level schedulers are also able to dynamically scale the concurrency permitted to each cloud (at least one study indicates that the concurrency depicted on one cloud vs an additional cloud should be predetermined according to price). As an example, when a workload can accept a bit higher latency, the system may buffer the request and combine them in order to complete the requests more efficiently (in terms of total authorised compute time spent).

An example is a service of image processing being executed on multi-cloud serverless [11]. Imagine that AWS is less expensive at image compression that is CPU-intensive (due to memory-based approach) and Google is less expensive at many tiny thumbnail generations (due to the free call count). The orchestrator can send big images to AWS Lambda and small projects to GCF thus cost is optimized in each category. The cost per request on each would be monitored and modified when something like Google comes up with a new discount or AWS has free tier capacity to spare.

To conclude, multi-cloud solutions create an opportunity to pursue the lowest cost at a given time. Nevertheless, that needs advanced surveillance and robotization. In the absence of it, running in multiple clouds may simply increase your cloud bill twice fold instead of reducing it half. The first steps the organizations should take are to review their workloads across each platform (such services and open-source tools exist to compare the performance of FaaS across the clouds) and deploy policies that direct traffic to the most optimal location. It involves intensive efficiency game: shaving a fraction of a cent per call, but in large volumes, or in cost-conscious applications, these efficiencies may amount to big differences.

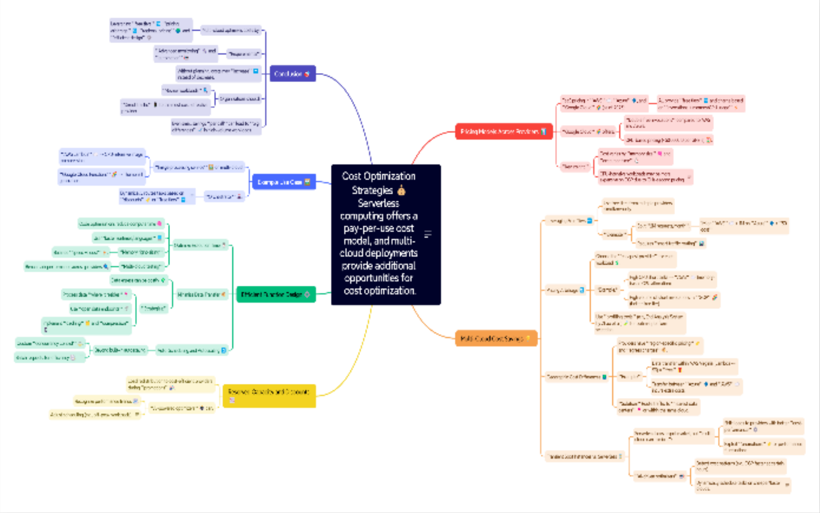

Cost Optimization Strategies 💰 Serverless computing offers a pay-per-use cost model, and multi-cloud deployments provide additional opportunities for cost optimization.
Conclusion
Example Use Case
Efficient Function Design
Pricing Models Across Providers
Multi-Cloud Cost Savings

Figure 6. Mind Map of Cost Optimization Strategies in Multi-Cloud Serverless Computing

4. Resilience and Fault-Tolerance

The potential of multi-cloud serverless computing has been set up in the context of resilience i.e. the capability to ensure operational services in the event of failures. This is because by using more cloud services based on various independent infrastructures, applications can enjoy the availability levels that could not be attained using one provider. In this case, we talk about high availability, disaster recovery, and failover in multi-cloud FaaS environments, the way they can be used in the context of mission-critical applications such as fintech.

4.1. High Availability via Redundancy

The ultimate aim in a multi-cloud configuration is to make sure that there is no single point of failure such as a cloud provider as well. This is normally achieved by the use of redundant deployments. Any critical operation is sent to two or more (the more the better) cloud operations, preferably dissimilar geographically. The system can be operated in an active-active mode, i.e., all the clouds carry out their functions actively serving requests or active-passive mode where one cloud is primary and others are hot standbys. The active-active option has a load distribution advantage, and may also lead to faster read throughput (both clouds handle traffic), whereas active-passive may be easier to implement (only failover on error). Banking communities and payment processors are turning to active-active multi-cloud when it comes to payment processing, as there can never be any kind of downtime. An example would be to have a payment API deployed in AWS in region X and Azure in region Y, in normal operation, load should be distributed between them (perhaps per region of the users, or even via round-robin). When one of them fails or slows down, the other supporting takes the entire burden [7].

4.2. Failover Mechanisms

The concept of providing failover implementation between clouds involves management of global traffic. It has some general methods:

- **DNS-based failover:** With health checks in a DNS service (e.g. AWS Route 53 or Cloudflare DNS), it is possible to assign multiple IPs or endpoints to a domain and configure an automatic failover. In the case of serverless this usually implies that each deployment to the cloud is served over an endpoint specific to the cloud (API Gateway URL etc), so DNS is pointed at the healthy one. Caching may cause DNS failover to require tens of seconds, but DNS failover is cloud-unsensitive and easy.
- **Anycast and Global Load Balancers:** Global load balancing is provided by some providers and third-parties and allows routing based on proximity and health to route traffic to alternate backends (including clouds). A simple example is that Google Cloud Load Balancing can be configured to target external endpoints or that a service such as Azure Front Door or Cloudflare Load Balancing can be used which will support health-probing endpoints in different clouds and use those in the region where the traffic performs best. The detection of an outage and rerouting can be achieved within a few seconds or so using these solutions.
- **Client-side Logic:** In other cases the client application (or an SDK) may be multi-cloud aware. As an example, a fintech mobile application might call the primary API endpoint to Cloud A then fall back to its secondary endpoint in Cloud B when calls to primary fail. This is the method of extreme failover and even more elaborate coding under clients, yet guarantees no reliance on third-party routing.

More importantly, failover must be stateful when required so dropped transactions can be avoided. When a request was processed by a function on Cloud A but it fails, can the same be resumed or retried by the one on Cloud B? It is imperative that idempotent function design (the same event reprocessed produces the same result) be used. A cross-platform messaging queue or datastore may help: as an example, an event may be pushed to a cloud-neutral queue (such as Kafka or RabbitMQ which is provisioned on independently owned infrastructure) and retrieved by either cloud. When Cloud A cuts out, the message is there and cloud B will process it. In this manner, the at-least-once delivery is preserved throughout failover.

4.3. Disaster Recovery (DR)

In addition to near-time failover, multi-cloud provides solid disaster recovery capability. Classical DR is the practice of having a secondary site which could be activated in case of catastrophic failure of the primary. Using multi-cloud FaaS, essentially you always have an instance of DR sites running. Yet, you should still expect the scenarios such as a mass outage of a whole cloud provider (or some important service within a cloud), and predict them. Another case in point was the AWS us-east-1 outage in December 2021 that knocked out numerous services; only single-tenant AWS companies went offline, and those that carried out cross-cloud deployments could easily shift to another provider. Multi-cloud DR planning is the process of:

- **Data Backup across Clouds:** Make sure that the data is duplicated or supported to the other cloud. In case you are using cloud specific databases (e.g. DynamoDB on AWS), you can stream the backups to an Azure Cosmos DB or a neutral storage. Certain fintech companies maintain duplicate copies of the transactional information in various clouds so that they can maintain operations across both sides.

- **Infrastructure as Code:** Use templates of infrastructure-as-code (Terraform, etc) so that your serverless environment could be replicated on a second cloud at any time. Again, you can have it even without active-active: The essential fact is that you can spin up your whole stack on Cloud B in a few minutes. The Serverless Framework or Terraform can send the same assemblage of functions to some other provider with some proper settings ready which is a DR deployment in the event that it is not in-use at outset.
- **Chaos Engineering and Drills:** Test cloud-to-cloud failover on a regular basis. Simulate out one provider and see what can happen 100 % with the other. This raised some interesting results in practice e.g. possibly Cloud B has a lower concurrency capping which does not manifest at 50% traffic but at 100%. Banks have been known to carry out game days where they deliberately disconnect one of the clouds but ensure that the processing of payments still works on the other as the regulators and internal risk officers require to see how they can withstand being put offline.

4.4. Distributed Consensus and Consistency

When dealing with a multi-cloud setting where capacity in different clouds might be involved in doing different components of a workflow, consistency of operations might be difficult to achieve. As an example, see a trading system wherein Cloud A runs trade orders and Cloud B refreshes account balances. In case there is failure of Cloud A after issuing some orders, then cloud B ought to be informed of what has been done so as to maintain balances. The solutions in this case could include a consensus mechanism or at minimum strong checkpoints. In many cases, architectures can be designed such that entirely each transaction is done within a single cloud (avoiding cross-cloud partially committing), and that eventual consistency is used to reconcile. It is a sophisticated matter and can only be mentioned here as a fault tolerance problem of not having a single cloud failure cause inconsistent state. In Practically, most multi-cloud serverless designs envisage simple ways of doing this: e.g. users in EU are always served by Cloud X (and served only by Cloud Y in exceptional cases), and users in US by Cloud Y (and served only by Cloud X in exceptional cases). This restrains the degree to which the clouds are entangled at a certain time thus ensuring less consistency problems.

Such principles have real-life examples in the fintech sector. Payments processors such as Form3 have openly talked about their multi-cloud resilience: they have their payment workloads distributed between AWS and GCP, such that workloads on one cloud can be shifted to the other when the first cloud has failed. They point out that authorities are now urgently promoting this strategy, the FCA in the UK, and DORA in the EU view multi-cloud as a means to avoid possible systemic outages. There is such a high price of payment downtime (cost and reputation wise) that the added expenditure on multi-cloud will pay off. The example of Form3 also demonstrates that multi-cloud could make things more scalable when under pressure: when a system is overloaded, two clouds means that it has more room to scale functions horizontally [7]. In short, multi-cloud serverless computing will achieve more resilience with redundancy and quick failover. This needs diligent engineering, the load balancing of global load, consistent replication of data and a complete testing. Well implemented, a multi-cloud system is tolerant of it cloud provider going out of business; one with very little effect the end users end-users a degree of fault tolerance that is becoming more and more required in transmitting important programs.

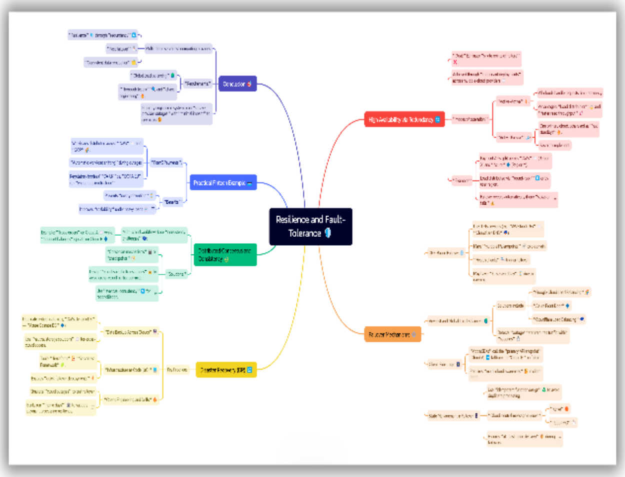

Resilience and Fault-Tolerance
Conclusion
Practical Fintech Example
Distributed Consensus and Consistency
Disaster Recovery (DR)
High Availability & Redundancy
Failover Mechanisms

Figure 7. Mind Map of Resilience and Fault-Tolerance in Multi-Cloud Serverless Computing

5. Case Studies / Examples (FinTech Focus)

To establish a reference point, we resort to the discussion of the application of multi-cloud serverless strategies (or their potential application) in the fintech area. Fintech solutions may have high requirements as far as reliability, low latency, security, and compliance are concerned, which makes multi-cloud FaaS architectures especially beneficial to fintech applications. The following are some examples and scenarios on how multi-cloud serverless affected fintech:

5.1. Global Payment Processing

Think of a credit card handling business dealing in transactions all over the world. The transactions need to be authorized within milliseconds and outages can cause commerce to stop. Through multi-cloud serverless design, the company spreads its authorization functionality on at least two clouds (AWS, GCP, Azure) and in more than two regions. An incoming transaction will send to the nearest the most possible cloud deployment. This minimizes latency (user in Asia calls an Asia-region function, on AWS or on Azure) and provides load distribution in a normal situation. In case a whole region of the cloud becomes disabled, transaction fails over to another region on the same provider, or on a different provider. To give an example, a multi-cloud setup would have auto referenced APAC transactions to Azure function in Singapore in November 2020 when an AWS region had a problem and sustained uptime. Such an approach is consistent with actual fintech procedures: the Form3 payment technology provider in the cloud notes that it focuses on multi-cloud specifically to prevent payment-related service disruption in case a single cloud provider experiences an outage. In their case, transactions do not get distorted but get diverted to another provider immediately. The outcome can almost be perfect 100 percent uptime something very important since payment downtimes not only provoke customer fury but also draw regulatory penalties [7].

5.2. Trading Platforms and Exchanges

Stock exchanges process huge amounts of orders, and must be able to withstand spikes (e.g. on market open, on volatile news). Multi-cloud serverless would fit well in the case of a trading firm using microservices to ingest trade orders, applying algorithms to them and confirm the trade. When these functions are distributed to different clouds they have increased throughput and fault tolerance. They may also use Google Cloud Functions to get its cold start speed and CPU power to quickly execute computationally demanding pricing algorithms and AWS Lambda to pull on their databases and send user notifications (as they live in the AWS ecosystem). In the event of slowdown of Google cloud, orders may be shifted to either AWS or Azure platform where they are awaiting a backup service.

Certain stock exchanges are actually looking at multi-cloud active/active relationships to ensure there is no down time during trading infrastructures regulators such as SEC and European have set contingencies in place to ensure there is reliable backing of exchange infrastructure (which is what they have been requesting). Serverless provides an additional advantage in such a situation; automatic scaling. During so-called high-volume days, two clouds which scale out entails essentially twice the capacity of one cloud. According to industry luminaries, financial institutions are ready to take up complexity as long as it enhances resilience in such situations when it comes to multicloud [8].

5.3. Regulatory Compliance & Data Residency

Take a case in point of a fintech firm that provides a digital banking solution in several nations. they are required to store EU customer data within EU data centers (GDPR) and UK data inside the UK, etc. They usually follow a multi-cloud approach where they implement the same serverless back ends to fulfill local compliance rules, i.e., use AWS to support EU and Azure to support UK assuming that they both have the necessary local regions or certifications. In so doing, they will be able to guarantee data sovereignty (no data goes outside its region / cloud) and yet be able to operate a single service. They are even able to change providers in an area should one fail to comply or change in regulations. It was observed when some banks were concerned about possible exposure to the U.S. CLOUD Act with certain providers using multi-cloud infrastructure gave them the ability to move sensitive workloads of the organizations to a new cloud in a new jurisdiction in case of necessity.

In practice, a significant number of banks adopt a multicloud strategy in part to ensure that they can demonstrate to regulators that they have a way out of using any particular cloud and do not rely too heavily on it (this is called avoiding cloud concentration risk). To take an example, we have been discussing how Goldman Sachs has shared a multi-cloud strategy where they have heavily deployed AWS, but they are also developing capability on Google Cloud, not purely as a best-of-breed service provider, but also as a matter of risk, to ensure they are never going to end up locked into one platform.

5.4. Open Banking and API Ecosystems:

The APIs (based on open banking) that enable third-party fintech apps to access bank services should be highly available and capable of variable loads. A bank can use serverless functions on cross clouds to implement its open banking interface. Developers that are not part of the company have only one API endpoint though on the backend, multiple clouds are employed. In case a specific cloud encounters API gateway emanation limits or failures, the routing layer of the bank sends the traffic to the functions of the alternative cloud. This will guarantee that there is uniform reliability to applications that use the APIs through fintech apps. Regarding costs, it can also be beneficial to the bank: in case one of the clouds is cheaper to make outbound API traffic, it can transfer most of the constant traffic there, with the other providing overflow to it during traffic spikes (similar to bursting clouds to the API). Such pattern was also suggested by some banking institutions in Asia and use Alibaba Cloud in some services and AWS in others with keep both poised in case of being called to provide APIs to fintech partners.

5.5. Fraud Detection and Risk Analytics

Most of the fintech companies depend on the machine learning models to detect fraud in real time. Transactions to check the fraud risk are scored by using serverless functions. With multi-cloud deployment of these scoring functions, the fintech can employ specialized services with each provider, e.g. TensorFlow models on Google Cloud Functions (with Google ML optimizations) and have a backup model almost concurrently on AWS Lambda. In case the ML infrastructure provided by one of them is slow (let us assume some unexpected update to Google effectively creates more cold start to ML libs), the system may be redirected to AWS. Training data can also be stored in various clouds (some in Azure Cosmos DB, some in Google BigQuery); multi-cloud functions can be used, so that data is worked on where it is stored to avoid transfer fees. This type of scenario was anticipated in the Sampe et al. (2021) research on transparency in multicloud, so that developers could invoke functions against data in any cloud with no redesign of the entire system.

5.6. Cost-Sensitive Batch Jobs

The example of a hypothetical case is a fintech company offering the customer reports on their spending activity at the end of each month. This is a non real-time-critical batch-job with the prospect of millions of users, which makes it a cost-issue. With multi-cloud, the fintech can identify on an instance-by-instance basis which provider is the cheapest at that moment - say One provider has a discount on FaaS similar to Spot Instances (it is not a thing, but suppose the providers have similar promotions or there is low contention at specific periods). They may complete a half of the report-generating functions on one cloud and the other half on a different cloud, and at practically any time, move dynamically, in case the cloud functions are performed faster (cheaper). Basically, multi-cloud deployment becomes a means of carrying out some sort of bidding on a batch. Although spot pricing does not occur in pure FaaS, the same effect is achieved in case one cloud provider charges more at a certain time of the day (such as providers of nighttime data processing at a lower price). This situation demonstrates that multi-cloud could make cost-efficiency optimal on non-urgent workload by continually choosing the path with the least costs.

These are some illustrations of the significant advantages of multi-cloud serverless in fintech:

- **Near-zero downtime:** Ensuring availability of services such as payments and trading has become constant, as the single-cloud reliance has been removed and services can now run at the high service level agreements, such as finance (usually 99.99% availability or higher).
- **Improved performance:** It correctly routes the users to the nearest or fastest cloud, denying them latency which is essential in trading (where by milliseconds count) and improves user trading experience in digital banking.
- **Compliance assurance:** The multi-cloud environments can be customized to the regulatory requirements of various markets, which grants fintechs the freedom to run their operations across locations in a lawful manner.
- **Cost management:** The margins of Fintech may be thin (e.g. payment processing is volume business) and reducing the cost through multi-cloud optimization can be source of competitiveness. Multi-cloud enables the exploitation of the best each cloud has to offer in one business process, including better prices.

A single case study, illustrating this in the real world, is particularly interesting: Capital One, one of the giant banks with the reputation of the technology consumption, has intensely applied serverless (mostly on AWS). Although multi-cloud is not publicly mentioned, it can be concluded that they may have thought through, or already deployed, backup capability to another cloud or on-prem to support vital processes. A second example is the stock trading app Robinhood that has experienced a well-known outage on a day of high trading volume.

There were post-mortem examinations which indicate that a stronger multi-cloud system could have avoided the entire collapse of trading on their platform. In summary, fintech examples are a good reason that makes multi-cloud serverless appealing: failure is expensive (financially and reputation), and regulation can make the assumption of failure to be, in practical terms, mandatory. Based on these illustrations, we realize multi-cloud FaaS as more than a hypothetical ideal but an applied design that many of the prominent finance-based companies are considering to make highly dependable and efficient.

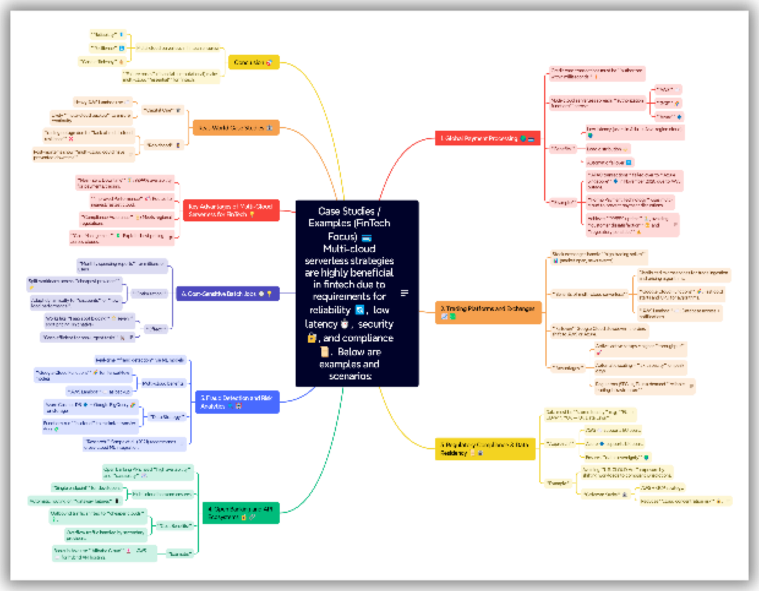

Case Studies / Examples (FinTech Focus) 💳 Multi-cloud serverless strategies are highly beneficial in fintech due to requirements for reliability 🔄, low latency ⏱, security 🔒, and compliance 📜. Below are examples and scenarios:
1. Global Payment Processing 🌍💳
2. Trading Platforms and Exchanges 📈💹
3. Fraud Detection and Risk Analytics 🔍🤖
4. Open Banking and API Ecosystems 🔓🔗
5. Regulatory Compliance & Data Residency 📜🔒
6. Cost-Sensitive Batch Jobs 💰💡
Key Advantages of Multi-Cloud Serverless for FinTech 💡
Real-World Case Studies

Figure 8. Mind Map of Resilience and Fault-Tolerance in Multi-Cloud Serverless Computing

6. Future Directions

The multi-cloud serverless computing is rapidly changing. A number of emerging trends and areas of research are on the verge of eliminating existing limitations and opening new possibilities, and multi-cloud FaaS will soon be more capable and convenient to work with:

6.1. Edge Serverless Computing

It is but logical to combine edge computing with multi-cloud. Edge serverless systems (e.g., Cloudflare Workers, Azure Edge Functions, AWS Lambda@Edge) enable the execution of code in at dozens or hundreds of edges near end-users. A possible vision of the future would be a network of serverless nodes distributed throughout central clouds and edge networks. Workloads would be able to work where it best fits: latenciesensitive components on the edge, intense computations on-cloud, and all that under a single multi-cloud umbrella. This is essentially an extension of multi-cloud to apply additionally to multi-tier (cloud + edge) deployments. As an example, a fintech IoT payment card may invoke a function on a nearby edge location to issue ultra-low latency approval, but may optionally fail-over to a cloud function in case of failure, or in more complex scenarios requiring more resources. Studies are already underway on how to harmonise the control of functions in cloud and on edge. This is where the idea of a serverless mesh architecture can be mentioned Malhotra et al. (2024): They suggest broadening service mesh technology to the edge sites and clouds by integrating serverless. This type of mesh would be able to dynamically dispatch the events either to an edge function or to a cloud function, and handle the networking between them, thereby offering a unified connectivity fabrics. The outcome would be an even more available and powerful system, perhaps even regional cloud failure could be damped by edge nodes accepting some slack.

6.2. AI-Driven Workload Optimization:

Manual tuning (of cost, performance, scaling) is not practical as the multi-cloud installations become more complex. We imagine that AI and machine learning will be of importance to autonomously optimize multi-cloud serverless systems. The orchestrators of the future may decide in real time on which cloud to their functions on, based on real time-based signals via reinforcement learning or predictive analytics: real-time prices (maybe one day there will be a market on FaaS spot pricing that allows a quick response to underlying operational costs), performance indicators, even energy consumption or carbon-footprint of a datacenter. Zhao et al. created an impression of having to note performance and cost needs in scheduling, and it is quite possible that sophisticated algorithms will be created on an on-going basis to learn and keep on updating multi-cloud scheduling policy. In addition, AI can assist with capacity planning that is predicting traffic and warming up functions on the cloud that is probably to be needed ahead of time or pre-provisioning. To some degree the multi-cloud orchestrator may be a self-driving system that performs optimization based on some specified objective (minimize cost subject to latency SLA, etc.) and requires no further human effort beyond initial configuration.

6.3. Enhanced Portability via WebAssembly (WASM)

WebAssembly is also becoming an interesting solution to serverless runtimes. WASM offers a sandboxed thin binary format executable whose execution can be uniform across environments. Researchers believe that it would help immensely to compile serverless functions to WASM to enhance portability and the cold start procedure. Theoretically a function that was compiled to WASM could run on any WASM runtime on a cloud or even inside a browser at the edge. This lessens the reliance to languages and run-time characteristic of individual clouds. It makes smaller deployment packages and quicker bootstrapping (WASM binaries are compact and fast to load). Serverless WASM is being made possible by a few open-source projects (such as the workerd runtime, of Cloudflare, or WASI projects). In a multi-cloud world of tomorrow, a dev could deploy a compile-once WASM module, which each provider runs in a compatible, compatible, runtime, a.k.a. write once, run everywhere. This would reduce present lock-in at the runtime level considerably and allow a more straightforward orchestration of cross-cloud execution of functions. It is also possible we will have cloud providers themselves providing bring your own runtime, though WASM, and that it becomes simpler to standardize the execution environment across clouds.

6.4. Open-Source and Interoperable FaaS Frameworks

To address the proprietary fragmentation, the industry will probably converge around open standards or frameworks to serverless. Such projects as Knative (which executes on top of Kubernetes) enable functions to execute on any Kubernetes cluster, i.e. any cloud or on-premises. A cloud-agnostic core-level multi-cloud serverless environment can also be built using Knative or a similar solution that allows companies to build such multi-cloud enabled serverless environment. Possibly, we will have Kubernetes-powered federated FaaS, e.g., a Kubernetes control plane that spans clouds and assigns the functions to various clusters (clouds) as required. CNCF CloudEvents standard is already useful in that it defines the event data format cross-service which will be useful in the migration of triggers between clouds. A third direction is workflow orchestration standards (such as Netflix Conductor or CNCF Argo Workflows) which may be used to orchestrate multi-step workflows at the serverless Workflow level across clouds in a provider-neutral manner. There is also potential for the development of something like the Serverless Application Model (SAM) or, in a far more minimal sense, Terraform modules to make multi-cloud configuration more declarative. Simply stated, the advocating of multi-cloud by design within frameworks will help developers implement these policies by far much easier with no need of custom solutions.

6.5. Improved Observability and Cross-Cloud Tooling

What we expect is more specific tools dealing with multi-cloud monitoring and debugging. Companies such as Dynatrace and Datadog are also supporting to intake telemetry across many clouds and give a single pane of view. In the future, tools could enable step-through debugging of a serverless workflow jumping between AWS and Azure, and this would be of immense help to the development. In fact, some research even suggest the use of a distributed ledger/blockchain to do logging so that there is an immutable, chronological log of events across clouds to analyze in a post-mortem attempt. Tooling in security will also evolve, e.g., the use of centralized policy engines (such as OPA - Open Policy Agent) to apply security policies consistently across clouds, e.g. that no functions are deployed in uncertified regions.

6.6. Serverless for Stateful and Long-Running Tasks

Serverless has been traditionally constrained to stateless, short tasks. This gap is being filled with emergent technologies such as serverless databases (e.g. Fauna, DynamoDB, etc.) and stateful function frameworks (e.g. Azure Durable Functions or the Netflix Stateful Functions on Flink). In multi-cloud, there is the difficulty in managing state which could be accessed by functions in different clouds. The concept of a global state layer, possibly supported by decentralized storage, or decentralized consensus mechanisms, that all functions can read/write to, with low latency, will be one future direction. This may be in form of distributed cache or database which is replicated across providers. There is another direction: long-running processes: a 30-minute process may be governed by an open-source FaaS framework, chaining ephemeral functions in different clouds (passing them to each other). State and coordination solution will increase the application space to the multi-cloud serverless at the risk of stepping on microservices VMs/containers.

6.7. Energy Efficiency and Green Computing

Multi-cloud serverless has the potential of even optimizing energy usage or carbon footprint as sustainability becomes an objective. Workloads could be directed in the future, by schedulers, to regions of the cloud that were less carbon-intensive at the time (some cloud providers already provide carbon data by region). This flexibility of Serverless, it can run anywhere, may mean it is possible to have a more intelligent “carbon-aware” deployment: e.g. run a batch in a data center using renewable energy now, pause or move when only coal-powered energy can be used. This is not certain, but follows general trends in cloud computing to make it less environmentally negative.

6.8. Economics and New Business Models:

Lastly, multi-cloud serverless could also enable new business models, i.e., third-party serverless brokers (the VSP concept) as a service. Prime factors that make companies use a multi-cloud orchestrator but not create their own out of scratch include taking out an orchestrator service subscription that offers a unified FaaS API on top of many clouds. Even smaller cloud or on-prem resources could be in the mix of that broker. Provided that such services reach maturity, it may induce larger cloud vendors to work on establishing interoperability (lest they be excluded in multi-cloud pools). Already there are startups developing such brokerage, which also originate as cloud-agnostic FaaS platforms. This may lead to commoditization of the FaaS layer, in the long term, that transfers its value to upper-level services.

To conclude, the multi-cloud serverless future is probably to be simpler, smarter, and stronger. Integration on the Edge will reduce latency; AI tuning will eke out in-efficiency; open standards, and the WASM, will eliminate most cloud incompatibilities. The developers might not even need to think in the "Cloud A vs Cloud B" whatsoever in the future- they will just code the functions, and they will be deployed and executed by the platform in the most optimal and suitable location, whether that is a cloud region or an edge node, with all the watchdogs and security systems in place. To realize such a vision, more research and partnership of academia, industry, and cloud providers are needed, but the momentum is on the right path until such time that an innovative technique currently (multi-cloud FaaS) will be an industry standard best practice in using cloud application designs that are resilient and scalable in the near future.

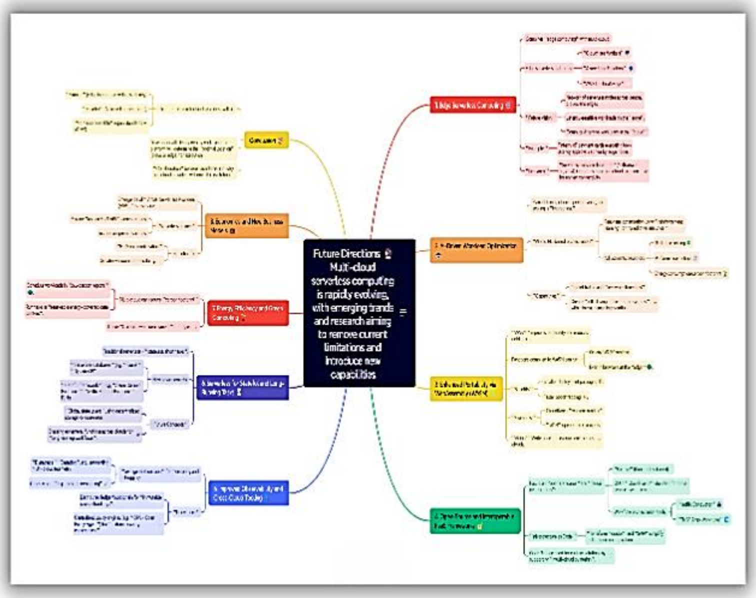

Future Directions
Multi-cloud serverless computing is rapidly evolving, with emerging trends and research aiming to remove current limitations and introduce new capabilities
Conclusion

Figure 9. Mind Map of Future Directions in Multi-Cloud Serverless Computing

7. Conclusion

Multi-cloud serverless computing is a very interesting approach to developing resilient and cost effective systems through using the power of many cloud providers. The current article considered the events that lead to a mitigation of vendor lock-in, fault tolerance, and cost-optimizing advantages when a multi-cloud FaaS architecture approach is adopted, which is becoming vital to industries such as fintech whose downtime and inefficient operation can be considered highly penalizing. As we started, we provided these background facts; serverless computing makes deployments easier, but in the past, serverless computing increased the problem of lock-in and introduced new obstacles to monitoring, performance (cold starts), and state management. Multi-cloud strategies are an encouraging prospect, and they also provide a method of getting out of those potholes since not all the workloads are placed on one platform. Using the architectural overview we were able to come up with some of the main components of a multi-cloud serverless platform such as orchestration layers which act as the broker between clouds and event bridges which are used to convey messages between clouds and a unified monitoring and identity system. Developers can use this architecture to manage multiple and heterogeneous cloud environments as a single logical compute, storage, and services pool. We emphasized the fact that the idea of implementing such system is not trivial anymore, such challenges as inconsistent environments, data gravity and operational complexity should be addressed.

The central challenges that we researched in depth demonstrated the fact that the multi-cloud serverless is not a magic bullet. In a multi-cloud scenario, problems of cold start latency, augmented observability complexity, security issues across multiple platforms and regulatory compliance of the system may become particularly difficult. They are current research and engineering fronts. We also talked of the ways to address them, including distributed tracing, and policy-as-code that extends to multi-cloud security, but we acknowledged that there are still gaps (such as the fact that debugging is still hard to do across clouds, and that may be a great application of these approaches). Conversely, in the opposite entry of the ledger, we learned more about cost optimization in multi-cloud. Through comparisons of pricing schemes, we demonstrated that an intelligent multi-cloud installation can be used to reduce costs: either via each provider offering a free tier of service, or via dynamic shifting of loads to lowest cost execution point, or even through the mere introduction of competition (clouds being pushed to offer better deals with the odd lots of the workload). The cost comparison table and examples indicate that the wiggle room to assign work may bring cost savings and avoid cost increase scenarios on a vendor basis translating up through the application.

We then moved to resilience and fault-tolerance, which is probably the primary explanation why those chase multi-cloud. Multi-cloud serverless systems can produce extremely high levels of availability. We outlined designs of active-active deployments and fast recovery which ensure that the services stay up even when there are failures which would immobilize the single-cloud system. The practical examples of fintech use can confirm that these trends are not merely hypothetical, but are gaining practical relevance, they are recommended by regulators, and necessitated by the demands of customers who demand to be served at all times. We also pointed out that multi-cloud mentality introduces a need to re-think disaster recovery (it is a constant situation, not an ability to switch, at a certain moment, in a certain direction) and design with caution (emerging failure modes should be considered, such as partial failure or cross-cloud inconsistency). We anchored these concepts in realize-able scenarios in our case studies, which discussed fintech: global payments, trading platforms, open banking APIs, and fraud detection. These cases demonstrated enhanced reliability (absence of a single point of failure), performance (its reduced latency provided through geo-distribution), and compliance (data residency as per jurisdiction based on the diversity of IaaS providers.). They were also used to demonstrate that multi-cloud is not the preserve of academia but inspired by serious business requirements. Fintech that tries to be on the cutting edge of the cloud usage because of its high failure costs sometimes predicts tendencies that can be common later to other industries (e.g., healthcare, Internet of Things).

If we turn to the future development directions, we noticed significant trends defining multi-cloud serverless computing in the next few years. Particularly, the merging between edge computing and multi-cloud will drive serverless to the next levels of low-latency calculation. These two advances of AI that uses intelligent scheduling and WebAssembly that provides portable deployment of functions will ease two bottlenecks of performance tuning and portability that are currently problematic. In addition, the open-source frameworks and standards would most probably introduce harmonies between service suppliers, and the multi-cloud implementation would not require custom engineering to be less challenging to engage. We envision in some way a slow abstraction where the developer does not have to care much anymore about the variety between clouds and where the complexity is taken over by the underlying platform (which may be a third party platform or an open framework). It will entail the issues that we listed above, such as intense multi-cloud observability, homogenized security, and cross-cloud data-management. This direction is indicated by early research e.g. serverless mesh concept and multi-cloud function benchmarking.

7.1. Research Gaps

Although multi-cloud serverless is an exciting future research topic, it has open research questions and practical gaps. One gap is in formal verification of multi-cloud workflows - ensuring that a process will behave the same regardless of the way it is split and where it executes (useful e.g. to comply with legal requirements and show that a process is correct). Some of it is economic, as in creating pricing functions or market protocols of carrying out functions and it may include bidding across clouds. Multi-cloud serverless is a new area, and we do not have standard benchmarks yet; much of the current benchmarking is dedicated to single-provider measurements of performance. Developing multi-cloud benchmark suites (including such use cases as failover time, multi-cloud latency, cost in various loading patterns) would greatly facilitate the process and the advancement by providers. Tooling is an area in which developers are not used to multi-cloud serverless: debugging, discussed earlier, is one example of that; another is the absence of simulation tools to understand how multi-cloud architectures will perform before implementing them. These are points that the academic and the industry can play a role perhaps by adding to existing serverless simulators or emulators that can support multi-cloud environments.

In conclusion, multi-cloud serverless computing and its variant called FaaS architecture present a high-powered strategy of creating the system that should not fail and should optimize costs. The practices and technologies will continue to evolve, but the trend is obvious: as the concept of cloud computing reaches maturity, the ease of using a combination of vendors will become a feature of sound, enterprise system design. Considering after microservices had become a de facto standard of scalable design, multi-cloud serverless can also become a de facto standard of reliable and effective cloud operation. The companies that manage and learn to use these architectures successfully (managing the related issues and challenges) will be best fit to provide continuous services and streamlined user experiences to navigate the constantly changing cloud environment.

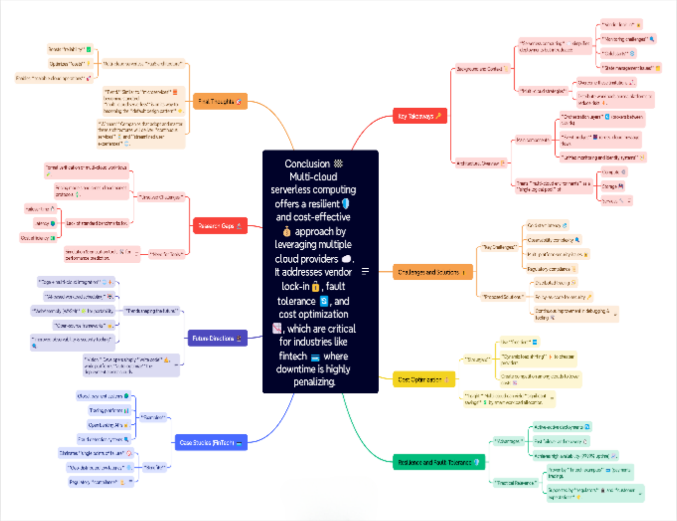


Figure 10. Mind Map of Conclusion and Key Takeaways in Multi-Cloud Serverless Computing

References

[1] Haidong Zhao; Zakaria Benomar; Tobias Pfandzelter; Nikolaos Georgantas. (2022). "Supporting Multi-Cloud in Serverless Computing. "Referred from https://ar5iv.labs.arxiv.org/html/2209.09367#:~:text=Serverless%20computing%20is%20a%20promising,1

[2] Ataollah Fatahi Baarzi (2021). "On Merits and Viability of Multi-Cloud Serverless." Referred from https://cirrus.ece.ubc.ca/papers/socc21_multicloud_serverless.pdf#:~:text=Virtual%20Serverless%20Provider%20,enhanced%20cost%2C%20performance%2C%20and%20scalability

[3] Josep Sampe; Pedro Garcia-Lopez; Marc Sanchez-Artigas; Gil Vernik; Pol Roca-Llaberia; Aitor Arjona. (2021). "Toward Multicloud Access Transparency in Serverless Computing." Referred from https://www.researchgate.net/publication/346172500_Toward_Multicloud_Access_Transparency_in_Serverless_Computing

[4] Wang, L. et al. (2018). "Peeking Behind the Curtains of Serverless Platforms." Referred from https://www.usenix.org/conference/atc18/presentation/wang-liang

[5] Hellerstein, J. M. et al. (2019). "Serverless Computing: One Step Forward, Two Steps Back." Referred from https://www.cidrdb.org/cidr2019/papers/p119-hellerstein-cidr19.pdf

[6] Castro, P. et al. (2022). "Hybrid Serverless Computing: Opportunities and Challenges." Referred from https://www.researchgate.net/publication/362568575_Hybrid_Serverless_Computing_Opportunities_and_Challenges

[7] Form3 (Tech Blog). (2025). "Building Resilience in Payments: Why multi-cloud is the future." Referred from https://www.form3.tech/news/payment-insights/why-multicloud-is-the-future

[8] CIO Dive – Ashare, M. (2025). "Banks diversify cloud portfolios to bolster resilience." Referred from https://www.ciodive.com/news/banking-hybrid-multicloud-strategy-generative-ai lseg/753445/#:~:text=global%20head%20of%20real%20time%2C,at%20LSEG%2C%20told%20CIO%20Dive

[9] Hardik Shah (Medium). (2024). "Serverless Computing Costs: A Deep Dive into AWS Lambda, Azure Functions, and Google Cloud Functions." Referred from https://hardiks.medium.com/serverless-computing-costs-a-deep-dive-into-aws-lambda-azure-functions-and-google-cloud-d797ca637b04

[10] Malhotra, S. et al. (2024). "Serverless Mesh Architectures for Multi-Cloud and Edge." Referred from https://philpapers.org/archive/SHUSMA-2.pdf#:~:text=and%20data%20con,Native

[11] Milvus AI Reference. (2023). "How does serverless architecture support multi-cloud deployments?" Referred from https://milvus.io/ai-quick-reference/how-does-serverless-architecture-support-multicloud-deployments

[12] Jonas, E. et al. (2019). "Cloud Programming Simplified: A Berkeley View on Serverless Computing." Referred from https://arxiv.org/abs/1902.03383

[13] Thirunagalingam, A. (2024). Transforming real-time data processing: the impact of AutoML on dynamic data

pipelines. Available at SSRN 5047601.

[14] Maroju, P. K. (2024). Advancing synergy of computing and artificial intelligence with innovations challenges and future prospects. FMDB Transactions on Sustainable Intelligent Networks, 1(1), 1-14.

[15] Aragani, Venu Madhav and Maroju, Praveen Kumar and Mudunuri, Lakshmi Narasimha Raju, "Efficient Distributed Training through Gradient Compression with Sparsification and Quantization Techniques" (September 29, 2021). Available at SSRN: https://ssrn.com/abstract=5022841 or http://dx.doi.org/10.2139/ssrn.5022841

[16] Sandeep Rangineni Latha Thamma reddi Sudheer Kumar Kothuru , Venkata Surendra Kumar, Anil Kumar Vadlamudi. Analysis on Data Engineering: Solving Data preparation tasks with ChatGPT to finish Data Preparation. Journal of Emerging Technologies and Innovative Research. 2023/12. (10)12, PP 11, https://www.jetir.org/view?paper=JETIR2312580

[17] B. C. C. Marella, "Streamlining Big Data Processing with Serverless Architectures for Efficient Analysis," FMDB Transactions on Sustainable Intelligent Networks., vol.1, no.4, pp. 242–251, 2024.

[18] Mohanarajesh Kommineni. (2022/9/30). Discover the Intersection Between AI and Robotics in Developing Autonomous Systems for Use in the Human World and Cloud Computing. International Numeric Journal of Machine Learning and Robots. 6. 1-19. Injmr. - 1