



SERVERLESS SERVICEMESH – A HIGH PERFORMANCE PROXY FOR SERVERLESS FUNCTIONS

Abstract

Agile software development space is quickly evolving by the day with greater adoption with clients as part of their digital transformation strategy. To aid swifter go-to-market for new products and services, deployment techniques such as canary releases and blue-green deployments have served the industry well however there are drawbacks with each approach. This paper explores the burgeoning use of service mesh as an emerging alternative that combines the pluses of the said approaches while doing away with the cons. It explores the trend at the vanguard of being able to deliver high quality capabilities with greater confidence to customers with speed and at scale. With the coming of the part – II of this white paper's edition – the paper aims to address sample implementation use cases and associated considerations.

2. Rising demand for faster frequent zero-defect releases

Organisations are now increasingly adopting microservices to address complex business challenges. Its ability to enable businesses with agility and react to changing needs of the customer and the volume of demand while enriching the insights gained in the process is only superseded by their innate nature of being launched quickly. Essential techniques aiding the swift launch of these services, mainly as part of capability uplift, include blue-green deployments and canary software release.

As for these deployment strategies, the benefits include straightforward roll-forward or roll-back in the case of blue-green and the incremental/step-wise availability of changes to the target audience, thereby limiting the blast radius should any roll-back decision be taken when the deployment option is exercised as a canary release. Either way – the ostensible cons with these approaches sometimes make them less attractive, i.e. cost of spawning a near production replica in the case of blue-green deployments or the hyper-care needed to test implementations in production through canary upgrades could take longer, mainly where manual verification is required.

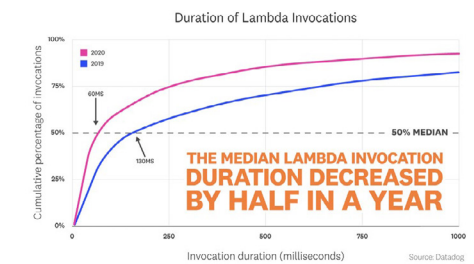
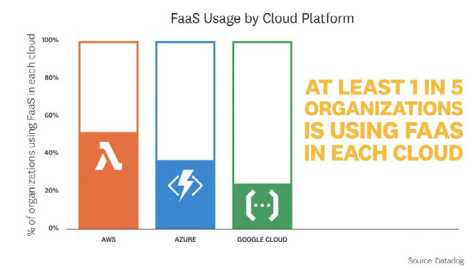
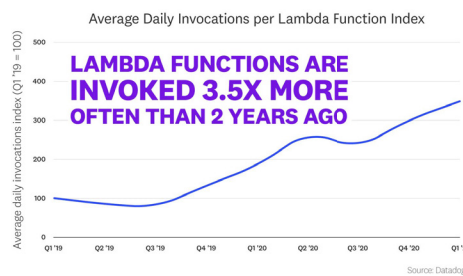
So, the question is – How can one address this gap? Or a better question would be – Is there a way to rollout functionality into production, run a failsafe QA exercise in the production environment thus ensuring high quality deliverables without having to run switchovers across multiple environments? We do have products such as istio for containerized applications however the options are lacking for functions-as-a-service apps.

3. Solution objective to support the growing popularity of FaaS

With the pay as you go model, the cost of hosting a serverless application can be orders of magnitude lower than any alternative approach. The other apparent benefit with FaaS is the burst scaling capability to handle load spikes with almost no notice. As called out in the Thundra blog (<https://blog.thundra.io/serverless-is-taking-off-heres-why-its-worth-hopping-on>), a dramatic shift is taking place. In 2018, less than 5% of enterprises were using serverless technology mainly in narrow tasks of a

specific nature however with corporations such as Telenor, Netflix, Reuters, AOL among others, serverless tech have registered consistent growth of 75% making it the fastest growing cloud service model.

The metrics & monitoring giant, datadog calls out in its recent publication - <https://www.datadoghq.com/state-of-serverless/> that Lambda functions are getting more popular, step functions (a serverless offering from AWS) is powering a vast variety of critical work-loads with more organisations adopting this framework.



To support this technology with the privileges extended to its Kubernetes equivalent, there is an immediate need to enable the deployment and test of FaaS components directly in production with minimal fuss and zero threat of breaking the existing application especially in brownfield applications.

For the solution to really stick with the developer community, it needs marry perfectly with the CI/CD infrastructure,

minimal cost footprint, reusable (preferably as a library) and in a plugin form that is external to their codebase regardless of development language (no code coupling). Addressing the afore mentioned NFRs, the solution must cater to specifics such as QA in the production environs for scenarios where FaaS services like Lambdas are updated to a newer version and bear the ability for FaaS services to invoke different versions of FaaS based on the presence of a header element.

4. Serverless service mesh approaches

With the pay as you go model, the cost of hosting a serverless application can be orders of magnitude lower than any alternative approach hence the incentive to address this requirement. Some of the solutions that were considered included the following

4.1. Code replication using versioning

With the pay as you go model, the top-of-mind solution happened to be that of creating a replica of the service carrying the updated version of the code. The deployment arrangement would be carried out through blue-green technique thus enabling quality assessment in the production environment.

Advantages include

- Simple solution and it does the trick

Disadvantages are

- Duplicating all serverless resources and keeping them running complicates the arrangement
- An inelegant & difficult to isolate issues as part of debugging
- Higher cost footprint
- Routing logic to the right function will have to be delegated to the API gateway which may not be the preferred pattern.

4.2. Differential routing using state machines (relevant to AWS step-functions/Azure logic apps with function connector)

The Step-functions capability is a serverless offering from AWS allows the developer flexibility to build conditional workflows to achieve business requirements. Azure offers something similar in logic apps with function connectors. These capabilities carry the feature that enables the state machine definition to conditionally route flows to different FaaS components (Functions/ lambdas etc.) i.e., production service or customized service (to be tested) based on the presence of a user-managed

header element passed through via the API gateway. With the help of templating engines such as cloud-formation, terraform – the entire arrangement can be built up or torn-down effortlessly.

Unlike the 4.1 solution, there is no replication/duplication of components required hence easier to implement, manage, debug, and comprehend. This

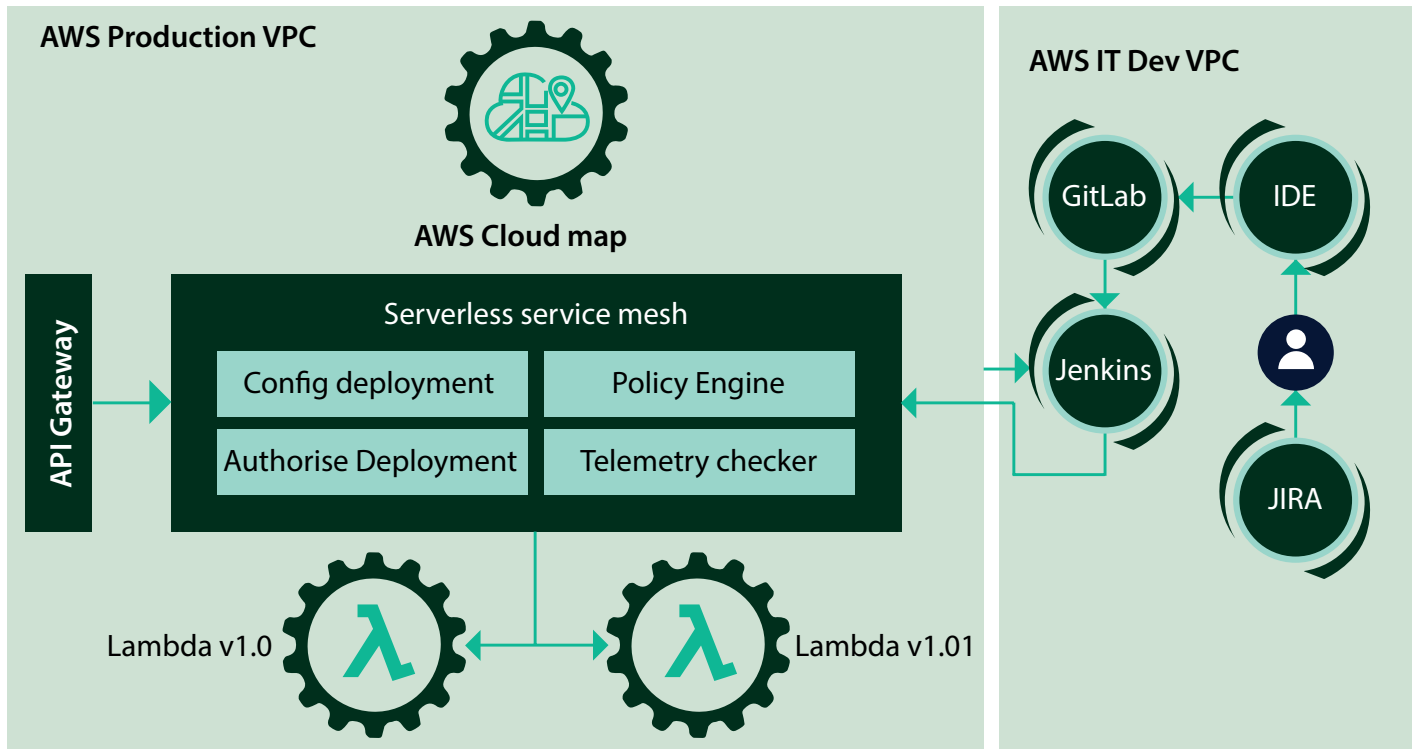
arrangement can be automatically setup/ torn-down using CI-CD tools suite.

However unlike the 4.1 solution, this option does increase cost from the greater number of transitions. It also isn't something that is reusable and needs to be built each time to a specific requirement therefore slowing down adoption.



4.3. Reusable routing library as a service mesh

The solution options thus far detail using native capabilities offered by the platforms however currently both in Azure and AWS, this capability lacks maturity. We envisage the creation of a library that functions as a routing layer on the serverless service (FaaS i.e., lambda/function/step-function etc). This layer takes instructions from an external configuration store such as a parameter store, database, and such. The CI-CD setup will essentially help modify the configuration entries as per the pipelines and flows designed.



Logical Architecture of routing library as a service mesh

4.3.1 How does this architecture work?

Firstly, the developer refers to the user story towards the change. He/she then builds the change and commits the change to the repo. The change modifies the serverless component (either lambda or stepfunction/lambda combination) resulting in minor version upgrade i.e., from v1.0 to v1.01. Subsequently the next set of steps follow through

- The build pipeline validates & verifies the change to the lambda service from v1.0 to v1.01

- Subsequently the cloudmap is updated with configuration elements associated with the new lambda version indicating any header elements, database, queue addresses that needs to be referenced with the new version. The cloudmap also carries a toggle against this lambda to help indicate whether the flow is in synthetic mode or in production mode. This flag influences the logging/monitoring/tracing elements of the flow as well to not corrupt any of the existing production logs.

On the side of deployment, the cloudmap is then considered by the CF script which

will then deploy any new infrastructure as dictated by the configuration against the lambda service in synthetic mode.

The serverless service mesh will wrap the lambda service helping route the traffic on the recommendation of the cloud map settings to the lambda service (v1.01) every time in the synthetic flow on the receipt of the request from the API g/w or any other service.

Once the mode is switched back into production post quality assurance, the CF script is triggered to redeploy the infrastructure as before and route the request back to the production instance v1.0

4.3.2 Flow to enable test

- a) Modify the flag entries and commit to repo
- b) Pipeline modifies the entries and then deploys the version to be tested.
- c) Any incoming request will have the header entry reflecting the settings from the flag in the param store/ database
- d) Post satisfactory testing, fire off the pipeline to promote the code to prod to deploy or if the flow needs to revert to the former state.

4.3.3 Merits & challenges

The advantages with this technique










- a) Clean solution with zero intervention from FaaS definition and that of deployment script
- b) No need to deploy any file during cutover. Simple updates to the config/param store will enable flows as needed
- c) Plug and play solution – generic & reusable
- d) Small cost footprint – two executions of the service in lieu of 1 for the duration of the test
- e) Easy adoption.

Practical challenges can be as follows

- a) If the teams use different runtimes i.e., Python, nodejs etc – this plugin libraries need to be managed separately



4.3.4 Comparison between serverless service mesh techniques.

Category	Option – 1 Version based Code-replication	Option – 2 Differential routing using state machines	Option – 3 Reusable routing library as a service mesh	Comments
Simplicity				Option 1 is simplest of the lot given that it replicates the entire environment as needed.
Cost				Option 2 does increase cost from the greater number of transitions. It also isn't something that is reusable (like option 3) and needs to be built each time to a specific requirement therefore slowing down adoption.
Applicability to other cloud providers				Options 1 & 3 are based on generic cloud concepts available with all service providers unlike option 2 which has certain specifics available only in AWS



5. Conclusion

In my practical experience, the approach using routing layer as a service mesh to help route requests thus enabling the release of high-quality services with confidence through apt levels of QA using production data to the extent possible especially for idempotent scenarios. In the next edition the paper will explore elements in sample implementation use-cases along with associated cost profile specifics & any additional considerations.



About the Author



Shreshta Shyamsundar
Senior Principal Technology Architect



For more information, contact askus@infosys.com



© 2022 Infosys Limited, Bengaluru, India. All Rights Reserved. Infosys believes the information in this document is accurate as of its publication date; such information is subject to change without notice. Infosys acknowledges the proprietary rights of other companies to the trademarks, product names and such other intellectual property rights mentioned in this document. Except as expressly permitted, neither this documentation nor any part of it may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, printing, photocopying, recording or otherwise, without the prior permission of Infosys Limited and/ or any named intellectual property rights holders under this document.