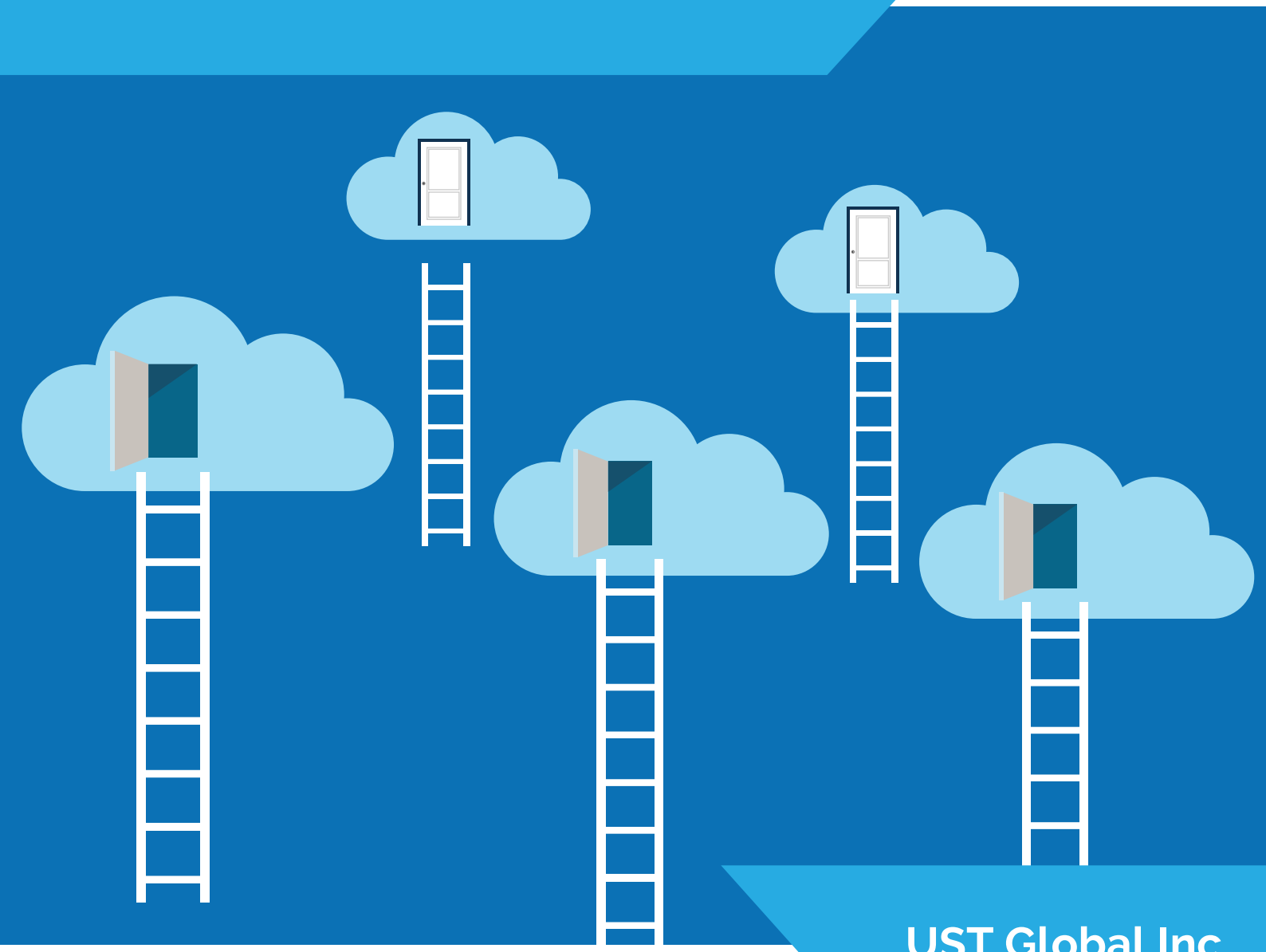
UST Global®

Serverless Development – Challenges and Opportunities



UST Global Inc,
March 2018

Saju Sankaran

CTO & Global Head of Cloud
UST Global

In today's digital world, technology is helping business to offer innovative solutions at reduced cost and improved agility like never before. Cloud is one such technology that has proven to provide rapid innovation and adapt at the speed of business with measurable benefits. The latest paradigm within cloud computing "Serverless Architecture" brings in the concept of "No Ops" a notch closer to reality. This architecture is quickly evolving and liberates the developers and application owners from the hassles of lifecycle management of underlying infrastructure but still meeting the availability, resiliency, and security of the services being developed.

Like any new service, development and deployment of Serverless within an environment especially in an enterprise need to follow an approach that is scalable, secure and extensible. This short article here explains, the personal experience of Deepu in working on Serverless environment. In the document, he explains the need for a framework such as Jazz (which he and his team developed and is already Open Sourced) and its evolution to simplify the life of a developer. He also provides few ideas on cost optimization, working within in the boundaries set by cloud service providers. This article also sets the foundation for organizations who are yet to start their journey on to the Serverless architecture and also provides optimization ideas for the early adopters.

Serverless Development - Challenges and Opportunities

This article describes the challenges that I faced and my observations while designing and developing the Jazz framework.

Jazz is a CI/CD automation framework at its core for Serverless (aka FaaS) services development and management. Jazz is continuously built on top of the existing Jazz framework, and currently supports the best known FaaS implementation from AWS, AWS Lambda. Jazz started as an internal initiative for T-mobile, and is now available on GitHub as an open source project.

You can check out Jazz at: https://github.com/tmobile/jazz

Through this article, I intend to describe the Jazz specific challenges encountered in my work, and not a complete set of challenges involved in Serverless Development in general.

Achieving Simplicity

Serverless means

- Absolutely no servers to provision and manage
- Ability to scale with usage
- Never paying for idle
- High availability and fault tolerance built in

Serverless simplifies the development and deployment of services with infrastructure resources that requires zero administration from development teams and your in-house infrastructure support teams.

What it means is that, as an application developer, I do not need to worry about spinning up servers, installing runtimes and dependencies, deploying my application and configuring it, and addressing other concerns like scaling, availability, etc. – which is kind of very cool! I have less components to manage, which means more time for me to focus on business logic, and lower operational costs.

Looking at how Jazz evolved over a period – at the time it initially started as a simple application with a few stateless microservices and a SPA UI app at the front-end, the entire process was superfast and very efficient in the beginning. This is a phase where I did not pay a lot of attention to cost and code optimizations. I developed AWS Lambda functions (just NodeJS code) and plugged it in the API gateway – I was ready to go!

As I started developing more complex features in Jazz such as multi-environment support, configuration driven approach, module support, etc., there was a need for creating an extensible framework which was easy to maintain, unit test and troubleshoot, and at the same time optimized in terms of costs and performance. An enterprise application is not just a set of microservices and SPA. It needs to be designed and architected properly, just like in traditional application development – to leverage FaaS simplicity and its pay-as-you-go cost benefits. It is very easy to end up paying more money than a traditional server based architecture if the application is not designed properly.

For example, even a trivial code optimization you make to improve the performance will have a direct impact on the operational costs. Based on the current charging scheme of all Serverless implementations, it only charges for the time your code gets executed (the granularity of it varies across vendors). If you optimize your lambda function to execute in 300ms down from 1 second, you get 70% savings in costs immediately! This signifies the importance of how your application and every component in it should be designed and perfected in a Serverless world. And this applies to every Serverless resource we use and leverage!!

You can click here to know more about the Serverless architectural patterns and best practices recommendations from AWS.

Achieving Security

Serverless is inherently multitenant, which means that a single instance of a running service is used to serve multiple clients (and that's how it's cost effective). This strategy poses problems with security, stability or even performances.

AWS has recommendations specifically for securing APIs and addressing issues related to multitenancy. For example, employing a multi-account strategy would provide isolation between different clients' services or even different environments of the same service. This is again very vendor specific, and so, by using Serverless, we give up control to these vendor implementations which can fail and be compromised.

You can check out the video from AWS reinvent sessions on how to implement authentication and authorization in your Serverless application.

As Jazz evolved, it needed to have its own security and ACL features. Designing and implementing an ACL for Jazz has been challenging for us, considering that we need to operate within the constraints of AWS specific implementations – such as API gateway and resource limit constraints at the account level.

Optimizing Cost

As I mentioned earlier, using FaaS could be more expensive than a traditional Server based architecture, if not designed properly and if you do not make your choices correctly. FaaS is not 'Serverless' in all aspects; in fact, we should be aware of where and how we deploy functions across regions, accounts, etc. to develop a performing cost-effective solution.

Designing a Serverless application which is secure and cost effective does require significant work than traditional application development.

AWS Infrastructure limits

This is another major challenge we currently face while developing Jazz. Every resource has a limit set. For example, for AWS Lambda, there is a limit set for the maximum number of concurrent instances that can be run in an account. The same applies for every Serverless resource being used, such as API gateways, S3 buckets, etc. and every managed service being leveraged.

Check out the AWS documents here to know more about AWS service limits.

Additionally, there are challenges in the AWS Serverless infrastructure itself, and it fails occasionally for reasons such as throttling of AWS resources, unknown internal resource limits, CLI bugs, etc. AWS technical support teams have been available to help us resolve the challenges. If the issue is related to service limits, AWS provides an option to bump up the limits at an additional cost.

Conclusion

Jazz has a very impressive roadmap - the ability to deploy services in multiple accounts or regions, support for Serverless platforms like Azure and Google functions, support for additional application templates, security features and ACLs for fine-grained access control, CLI support, etc. are coming!

I expect all the challenges mentioned above to be resolved as Serverless architectures mature further. Open-source based portable FaaS implementations such as Apache OpenWhisk, Kubeless, etc. are gaining momentum in the Serverless space. I would expect to see some standardization around these Serverless platforms in the future. It would be cool to be able to plug in different integration components such as API gateways, Apigee, S3, Kinesis or others based on vendor platform capability or specific use cases, without modifying my code or developing more abstraction layers.

To know more about Jazz framework, please check out the below links and give it a try! Please do not forget to send us your feedback.

https://github.com/tmobile/jazz
https://github.com/tmobile/jazz/wiki

Deepu Sundaresan
UST Global Inc

ABOUT UST GLOBAL®

UST Global® is a fast-growing digital technology company that provides advanced computing and digital services to large private and public enterprises around the world. Driven by a larger purpose of Transforming Lives and the philosophy of "fewer Clients, more Attention", we bring in the entrepreneurial spirit that seeks the fastest path to value in today's digital economy. Our innovative technology services and pioneering social programs make us stand apart.

UST Global is headquartered in Aliso Viejo, California and operates in 21 countries. Our clients include Fortune 500 companies in Banking and Financial Services, Healthcare, Insurance, Retail, High Technology, Manufacturing, Shipping, and Telecom. UST Global believes in building long-lasting, strategic business relationships through agile and client-centric global engagement models that combine local experts and resources with cost, scale, and quality advantages of global operations.

For more information, please visit: www.ust-global.com

Corporate Office: UST Global ®
5 Polaris Way, Second Floor, Aliso Viejo, CA 92656
Tel: (949) 716-8757 Fax: (949) 716-8396
www.ust-global.com
For further information contact: info@ust-global.com


/USTGlobal


/USTGlobal

/ustglobalweb


/company/ust-global