Digitalizing a Monolithic Application

Boddam Linga Reddy

Principal Engineer
4424 134th pl se, Bellevue, WA, USA-98006

Abstract

The challenge to Retail Services Platform (RSP) which resides in a Java monolith software with over 400 SOAP operations running on WebLogic is demanding. As systems age, the development tools, hosting technology, and even system architecture that the product was built on, have become increasingly inextensible. The application attracts over 70 million transactions per day. Many cloud computing types were evaluated, such as PaaS (Platform as a Service), SaaS (Software as a Service) and IaaS (Infrastructure as a Service). After thorough research, PCF PaaS (Pivotal Cloud Foundry) was chosen, as all ecosystems in T-Mobile are On-Prem. Micro apps were built which retained compatibility between existing SOAP web service WSDL contract and calling clients. The combination of Hystrix and cloud configuration server avoided network latency, enabled resilience patterns, auto recovery and changing of application properties on-demand. This method; "Digitalizing a Monolithic Application" (DMA) was later adopted by several other technical groups in T-Mobile and was an inspiration to other international companies

Keywords: *Cloud, Monolithic*

INTRODUCTION

T-Mobile US [1] provides wireless voice, messaging, and data services in the United States, Puerto Rico, and the U.S. Virgin Islands under the T-Mobile and Metro by T-Mobile brands. The company operates as the third largest wireless network in the U.S. market with over 83 million customers and annual revenues over $34 billion. Its nationwide network reaches 98 percent of Americans through its EDGE 2G/HSPA 3G/HSPA+ 4G/4G LTE networks, as well as, through roaming agreements.

Client-Server technology [2] was employed in T-Mobile until 2006. This pattern exposed many disadvantages which are not limited to the need of a specialist operating system, high maintenance costs, additional manpower and disruption in case of network failures. RSP was the solution to address the issues.

Retail services layer is the strategic platform, and the foundation for next generation of User Interfaces, IHAPS (In House Application systems), Organic and Inorganic stores (Apple, Wal-Mart, Best buy, Costco, etc.), e-commerce shops Amazon, NewEgg, Venicom, salesforce etc. Business functionality was published, discovered, and consumed as part of a business ecosystem of network-aware and reusable technical services. Retail Services Platform is a robust solution which will allow T-Mobile to meet existing and future business needs in a rapid and effective manner.

THE CHALLENGE

The Retail Service Platform layer interacts with over 40 different front end and over 30 backend ecosystems and runs on a single code base. This layer absorbs 79% of the total real time purchase path transactions from shopping applications; Retail Web, Care, Self-care and TFB (T-Mobile For Business) channels. The layer also interacts with business-critical eco-systems such as payment and finance. RSP was formed with an intent to be light weight, however, Ever-growing business needs and the time-to-market requirements made this a very tightly coupled ecosystem.

Technical teams at T-Mobile were at the crossroads of either enhancing the layer, Or, introducing inevitable customer and system pain points. To release any new feature or functionality, the system had to take a multi-hour outage and bring down all customer-critical applications. The universal pill of 'recycling' the application was also taking hours, in the case of issues.

The system was built on SOA architecture principles [3] and the current age is over 10 years. Auto scaling is not possible and not entirely compatible with 'Dev-Ops' strategy. Build and deployment times are high as the entire code is required to be built and shipped. System and regression testing landscape ended up having more than 10,000 test cases in its entirety.

Due to overgrown technical orchestrations between the service layer and multiple backend systems, Response time was increased, and performance was decreased. Infrastructure cost rise was a continuous trend with the need to apply security patches over multiple servers. Telemetry (1) and Logging mechanisms were a challenge.

Lessons Learned: The finer level details during the engagement of a monolithic system should be given proper care, before its introduction into the landscape, given the fast-changing business and technical needs.

(1) ***Telemetry** is an automated communications process by which measurements and other data are collected at remote or inaccessible points and transmitted to receiving systems*

(splunk or in-house logging systems) for monitoring.

DISCUSSIONS

PCF [4] PaaS was elected to build micro-apps to overcome existing challenges in the monolithic application. The apps are built using 12 factor cloud-native principles [5]. Strangler pattern [6] was employed in DMA. The most important reason to consider a strangler application over a cut-over rewrite is reduced risk. Strangler pattern gives value steadily and the frequent releases allows better monitoring controls.

Apache CXF [7] and Spring Boot frameworks [8] were used to retain the existing SOAP WSDL [9] contracts. Simplified queuing mechanisms were built using RabbitMQ [10] and Kafka [11]. Several RabbitMQ server nodes on a local network are clustered together, forming a single logical broker. Queues are mirrored across several machines in a cluster, ensuring that even in the event of hardware failure, messages are safe. Kafka handles high-velocity and high-volume data. Also, enabled message throughput of thousands of messages per second. Kafka handles these messages with the very low latency of the range of milliseconds, demanded by most of the new use cases. Fault tolerance is an inherent capability in Kafka, to be resistant to node/machine failure within a cluster. Engagement of these frameworks and message queues yielded tremendous results.

To enable stronger resilience patterns and auto app recovery, Hystrix open source framework [12] from Netflix, was used. Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.

MongoDB NoSQL [13] was used to house 'catalog data'(2) which feeds Device, Plan and default financial data into multiple front end systems. The retrieval mechanism was highly efficient and proved a game-changer in purchase paths across retail and non-retail channels.

Continuous Integration and Continuous Development, CI/CD [14] pipeline was achieved using Jenkins and integrated into Bitbucket [15]. Newman Automation tool [16] was used for Unit and Integration testing. This empowered quality engineering team to run continuous test cycles. Splunk tool [17] was used for logging and troubleshooting. Telemetry was achieved using AppDynamics [18].

THE VISUAL

The high-level visual Figure 1, including the technology stack, used to achieve 'Digitalization of a Monolithic Application' is given below;

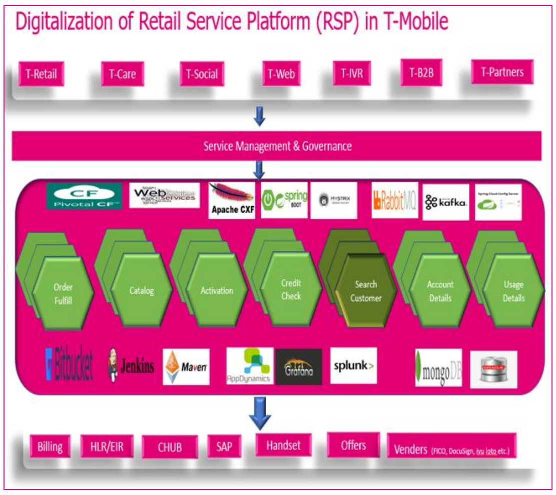

Figure 1

Retail Services Platform application was divided into micro-apps based on hey business functionality. The primary apps being; Customer, Order, Catalog, Offers, Credit Check, Activation, and Usage. The decimation of these functionalities empowered scalability and ease of operation across the enterprise.

Figure 2 The journey started in March 2017 (Blue line) which denotes traffic ramp down from Monolith and in parallel traffic was ramped up (Red line) using strangler pattern. No risks were identified during the transition and existing business flows and processes were not impacted.

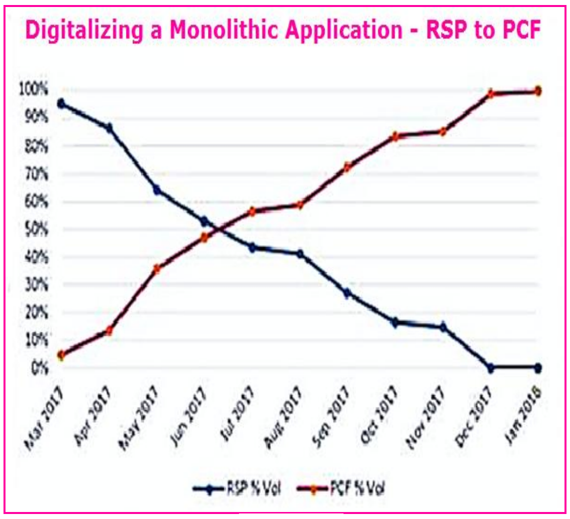

Figure 2

(2) ***Catalog Data*** *is a menu which contains all Devices (Phones, Accessories), Rate Plans, Pricing Information, Finance options, Product specific promotions and offers. This forms the core of shopping experience in any customer or representative facing channels in T-Mobile.*

CONCLUSION

The idea to not impact existing customer facing applications was attractive. The legacy applications can continue to call the same services and still achieve the business and enterprise goal of moving to a new digitalized service platform. The software's embedded in monolithic application dismantled with small manageable components. The CXF framework and spring boot works together spans the transaction thread and process it. RabbitMQ and Kafka message models decouples the JMS embedded in the WebLogic with greater through put. Hystrix enabled system resilience to relieve system or platform back pressure propagation. NOSQL Mongo Db made system high availability data inserts and retrieval. Automation on deployment process improved by leveraging cloud foundry blue, green, canary deployments without manual intervention. Operational excellency achieved by utilizing cloud foundry autoscaling, telemetry, pro-active alerts, real time AppDynamics displays.

RESULTS

Better response times Figure 3 (43% benefit) 400 milliseconds to 220 milliseconds. This was a huge win for our customers and IHAPS (In House Applications).

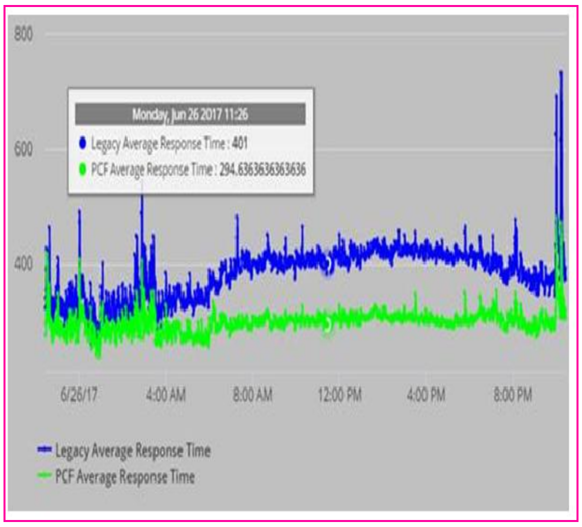

Figure 3

2. Zero downtime and daytime deployments: What used to be outage dependent system to launch business projects, had become 'Outage Free' as a result of deployment mechanism used in this transformation.

3. Fewer incidents (18 in 2016 versus 3 in 2017) and faster resolution of incidents (112 mins instead of 343 mins). A pure Dev-Ops strategy was achieved due to micro apps.

4. Scalability was a huge win as our Operations team did not have to spend tedious hours to add VM's and deploy applications.

5.Low cost in comparison to legacy system creation.

6.Improved Telemetry and Monitoring mechanism ensured alerts were triggered in real-time, thereby decreasing risk to business

7. Digital Death Star is pretty. Figure 4 Many apps were digitalized and monitored. The application flow map shown below, indicates the digital nature of the entire application.

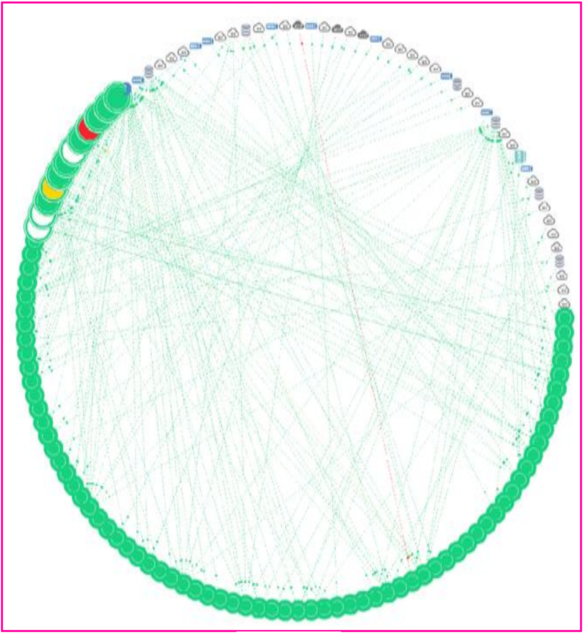

Figure 4

ACKNOWLEDGMENT

It takes a team to build a village, and the team was nothing but exceptional. Special Thanks to Darron Webb and Kris Wilson for allowing the team experiment an ideology and making it work!

Thanks to invaluable support from our leadership team; Cody Sanford, Chuck Knostman and Warren McNeel.

ABBREVIATIONS

PCF	Pivotal Cloud Foundry
RSP	Retail Service Platform
CI/CD	Continuous Integration/Continuous Delivery
PaaS	Platform as a Service
DAM	Digitalizing A Monolithic Application
LTM	Local Traffic Manager
SQL	Structured Query Language
SOAP	Simple Object Protocol

REFERENCES

[1] T-Mobile USA, INC. (2019). About T-Mobile. Retrieved from https://www.t-mobile.com/about-us

[2] CLIENT SERVER TECHNOLOGY SET TO ENHANCE ARMY TACTICAL C2 SYSTEMS. (1996). Inside the Army, 8(45), 4-4. Retrieved from http://www.jstor.org/stable/43979602

[3] The open Group. (2016). SOA architecture principles. Retrieved from: http://www.opengroup.org/soa/source-book/soa_refarch/p3.htm

[4] Pivotal Software Inc. (2019). PCF Pivotal Cloud Foundry. Retrieved from https://pivotal.io/platform

[5] Adam Wiggs. (2012). The Twelve-Factor App. Retrieved from https://github.com/heroku/12factor.

[6] Martin Fowler. (2014). StranglerApplication. https://martinfowler.com/bliki/StranglerFigApplication.html

[7] Microsoft Inc. (2019). Strangler Pattern. Retrieved from https://docs.microsoft.com/en-us/azure/architecture/patterns/strangler

[8] Apache CXF. (2019). Apache CX: Retrieved from http://cxf.apache.org/index.html

[9] Pivotal Software Inc. (2019). Sprint Boot: Retrieved https://spring.io/projects/spring-boot

[10] w3schools.com. (2019) WSDL: Retrieved from https://www.w3schools.com/xml/xml_wsdl.asp

[11] Pivotal Software Inc. (2019). RabbitMQ: Retrieved from https://www.rabbitmq.com/

[12] Apache Software Foundation. (2017). Kafka. Retrieved from https://kafka.apache.org/

[13] GitHub Inc. (2019). Hystrix: Retrieved from https://github.com/Netflix/Hystrix

[14] MongoDB Inc. (2019). MongoDB: Retrieved from https://www.mongodb.com/nosql-inline

[15] Marko Anastasov. 2019. CI/CD: Retrieved from: https://semaphoreci.com/blog/cicd-pipeline

[16] Atlassian. (2019). Bitbucket. Retrieved from https://bitbucket.org/product/features

[17] GitHub Inc. (2019). Newman: Retrieved from https://github.com/postmanlabs/newman

[18] Splunk Inc. (2019). Splunk: Retrieved from https://www.splunk.com/

[19] AppDynamics. (2019). AppDynamics: https://www.appdynamics.com/