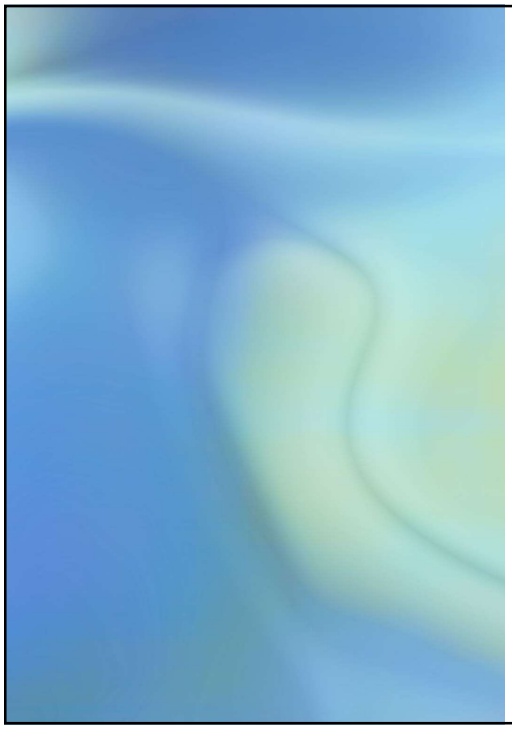

Microservice Architecture Enables DevOps

Raj Deol
Shru
Madhuri

Outline

- Paper Overview
- Background/Related Work
- Summary of Microservices Architecture
- Key Findings from the Paper
- Author's Evaluation
- Author's Conclusion
- Critique: Strengths/Weaknesses
- Critique: Evaluation
- Gaps & Challenges to Microservices

Paper Overview

- Bactory experience and lessons learned for migrating to Microservices Architecture
  - Backtory: Mobile Backend as a Service
- Backtory Company Problem
  - Needed an agile way to grow and scale the product meet growing demands and speed
  - Current toolsets were 'monolithic', slow to adapt, and were not cloud native
  - Changes to products took lot of time; they were looking for a continuous delivery model
  - Realization to change the architecture when faced with need to add chat as a service
- Hypothesis/Solution
  - Backtory envisioned rearchitecting their product using Microservices architecture
  - Planned an incremental migration approach to rearchitect product and delivery model
  - Lot of success but also lot of lessons learned

Related Work

- Microservices architecture has become an industry norm.
- Even though new companies may not start with this approach, they eventually migrate to make their business scale to growing demands
- Uber Case Study – Starting Architecture
  - Started with monolithic architecture approach for single offering in single city
  - Expansion to other cities and growing demand showed limitation of monolithic approach
  - Each improvement required rebuild, redeployment, and testing of entire application
  - REST API for passenger-driver connect, 3 adapters, MySQL db
- Uber Case Study – Migrated to Microservices Architecture
  - Migrated to multiple codebases to form microservice architecture
  - Introduced API gateway through which all drivers/passengers are connected
  - Different microservice for billing, notification, payments, driver mgmt., passenger mgmt.
  - Each microservices can scale independently with its own deployment and testing

Summary of Microservices Architecture

- Microservices is a cloud native architecture style that aims to realize software systems as a package of small services.
  - Each service is organized around business capabilities and by small team
  - Independently deployable and highly maintainable and testable
  - Loosely coupled – can run in it's own process while communicating through REST
- Not a new concept; most organizations have already adopted or in process of adopting it beyond a product to company wide
- Not a 'silver-bullet' - has potential drawbacks
  - Can be higher cost
  - Difficulty of Integration
  - Increased latency due to high volume of remote calls
  - Difficult to debug and test whole system

Key Findings from Paper

- Microservices and DevOps popularity started growing in 2012 and are one of the most searched technology keywords from 2013-2020.
- Microservices helped Backtory in shipping new features more frequently and providing scalability for the collective set of users
- After migration, deployment in the development environment was difficult
  - Although isolated services, there were many dependencies across services
- Distributed-system development required highly skilled developed
  - Despite lot of training, many developers struggled
- While benefit of each service designed autonomously sounds great (e.g. separate programming languages), too many differences creates chaos to manage
- Overall great benefit to Backtory as their system needed flexibility and they had the right tools and skills (Spring Cloud and Netflix OSS), which made migration/development easy

Backtory's Change to Delivery Model

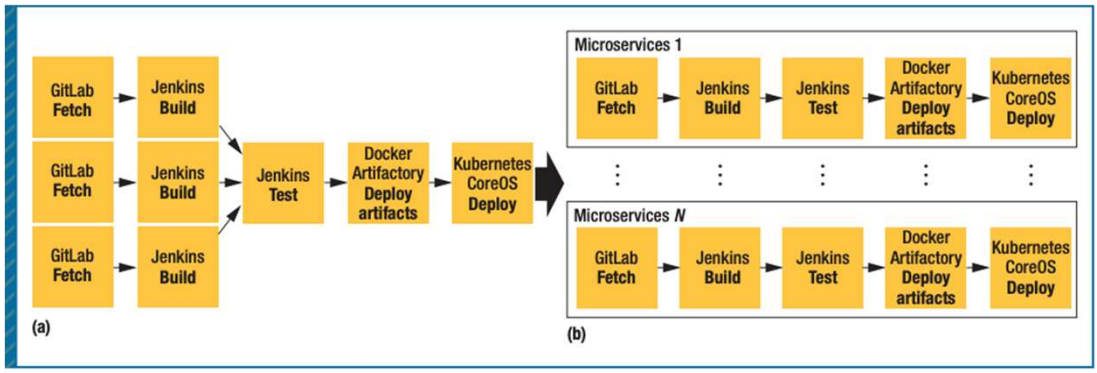

FIGURE 3. Moving from (a) a monolithic pipeline to (b) a microservices pipeline. The final delivery pipeline has independent delivery for each service, so each can be deployed independently.

Microservices Critique: Strengths/Weaknesses

- Strengths
  - Without doubt, Microservices helps make the software product be flexible and scalable
  - Enables DevOps to bridge gap between development and operations
  - Enables organizing by business capabilities – maximizes focus to specific customer product
  - Cloud native approach and take advantage of latest technologies like Containers
- Weaknesses
  - Requires up front planning & design (e.g. capabilities, integration, tools selection)
  - While easy to build individual service, lot of complexity in overall distributed system
  - High degree of skills in architecture planning and service development
  - Need well thought out migration plan, otherwise high change of failed effort

Paper Evaluation

- Positives
  - Paper presented Microservices in context of a real world example
  - Covered both technical architecture components and DevOps process
  - Covered both benefits and drawbacks of microservices
  - Clarified technologies used (Netflix OOS, Docker, Maven, Oracle, Spring, etc.)
  - Detailed the migration pattern for adopting Microservices
- Negatives
  - Small company case study with not lot of products
  - Did not describe why the final architecture had lot of dependencies
  - Technical descriptions/diagrams were confusing and hard to understand
  - Did not describe specifics of how data was separated or kept together
  - The article makes heavy emphasis on Netflix OOS and Spring for Microservices
    - In reality, microservices should be technology agnostic

GAPS

- Article described Microservices in context of re-architecting full product
  - Couldn't Backtory take incremental approach – i.e. start microservices only for new feature?
- Are there specific technologies that help adopt Microservices faster?
- What if there is lot of data integration between products? What are some cases where data can be fully separated versus kept as shared?
- If every team is making enhancements independently, how to manage compatibility when one product is behind and others are ahead?
- How to quantify cost and benefits for Microservices?

Questions?