Migrating to Cloud - Native Architectures Using Microservices: An Experience Report

Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi
Sharif University of Technology, Tehran, Iran - 2015

Sonam Gupta
Srinidhi Ramadurai
Smruthi Sridhar

Agenda

- Paper overview
- Introduction
- Background/Related Work
- Summary of New Technology Benchmark
- Key Contributions
- Authors Evaluations
- Conclusions
- Critique: Strength
- Critique: Weakness
- Critique: Evaluation
- Future Work

Paper overview

- Migration of an application named SSaaS (Server Side as a Service) in PegahTech Co.
- Specific to this project
- Lessons Learnt
- Challenges
- Sections
  - Background behind Microservice Architecture
  - SSaaS existing architecture & Target Architecture after Migration
  - Migration plan & steps followed
  - Lessons Learnt
  - Conclusion

Introduction

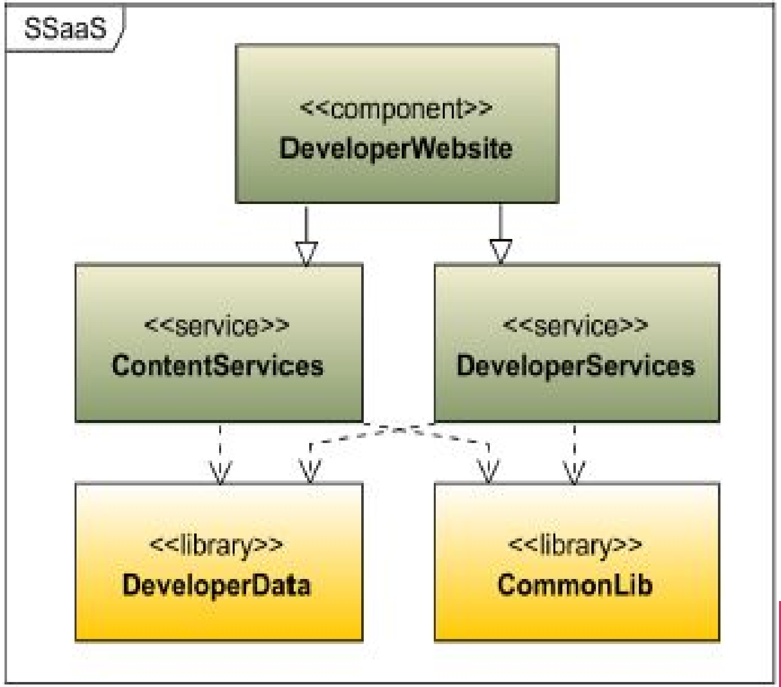
- Microservices:- Continuous Delivery & DevOps
- Continuous Delivery:- Continuous Integration & Continuous Deployment
- DevOps - Collaboration between Developers and Operations team
- Microservices Architecture Components:
  - Configuration Server
  - Service Discovery
  - Load Balancer
  - Circuit Breaker
  - Edge Server

Background

- SSaaS - server side programming part of their applications without knowing any server side languages
- First functionality - RDBMS as a service
- Future - Chat as a Service, Indexing as a Service, NoSQL as a Service
- Technology stack
  - Java using spring framework
  - Oracle -11
  - dependencies-Maven
  - Deployment-Jetty plugin
  - Repositories -Git.

Existing Architecture

- Common Lib
- DeveloperData
- DeveloperServices
- ContentServices
- DeveloperWebsite



Reasons to Migrate to Microservices

Requirement for Chat as a service: On demand capability

- Need for reusability
- Need for decentralized data governance
- Need for automated deployment
- Need for built-in scalability

Summary of New technology Stack

Components of New Technology Stack:

- Java Spring Boot
- Netflix OSS-Microservice specific components
- Eureka for ServiceDiscovery
- Ribbon for Load Balancer(Internal Load balancer)
- Hystrix -Circuit Breaker
- Zuul-Edge Server.

Monolith to MicroService -Related Concepts

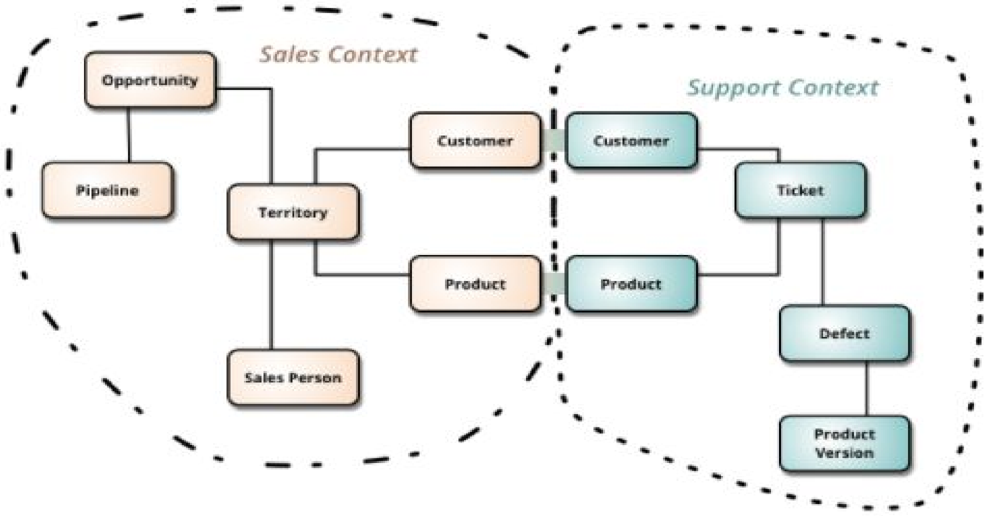
- Domain Driven Design
- Bounded Context

Domain Driven Design

- Supports the structuring of larger systems according to domains
- Each microservice is meant to constitute a domain
- only one microservice has to be changed in order to implement changes or to introduce new features.
- **Example-**Promotion service from Monolith e-commerce System

Bounded Context

- Divides the large models into different Bounded Contexts
- explicit about their interrelationships.



Re Architecting Logic

- Existing system is less Complex
- Re Architected system based on domain of Developer Data
- Put every set of cohesive entities into a service, such that the only one which can create and update that entity would be that service

Example:

Chat Services service could update or create the chat metadata entities.

Features of Target Architecture

- Chat Services service handle its metadata by itself ,not inside Developers Data.
- Introduce a new Resource Manager service in order to reserve resources-Oracle is moved from Devlopersdata to this service
- A new service to handle developer's information and its registered services.
- Transforming Developer Data from a library to a service.

Steps involved in Migration:

Step 1: Preparing the Continuous Integration Pipeline

- Allows developers to integrate their work with the others' early and often,and helps to prevent future conflicts
- As no of services increase while shifting to microservice architecture.No of instances running and deploying increases
- Virtualisation-less effective and costly
- Containerisation -deploy with low overhead and in isolation
- Deploy anywhere where containers are supported without changes to code or images

Docker

- Tool for containers.
- Pool of ready to use images in DockerHub
- Can be pulled and customised based on users needs
- Docker Registry -let organizations to have a private docker image repository

  Jenkins 9 - CI server.

self-hosted Gitlab -code repository.

Artifactory 11 as the artifact repository.

Step 2-Transforming Developer Data to a Service

Introducing Continuous Delivery

STEP-3:

- Separate source code, configuration, the environment specification to evolve independently
- Ability to change configuration without redeploying the source code.
- Docker removed the need for specifying environments since the Docker images produce the same behavior in different environments.
- separated services' code repositories to have a clearer change history and to separate the build life-cycle of each service.
- Automated deployment on a single server.

Introducing Edge Server

STEP 4:

To minimize the impact of internal changes on end-users

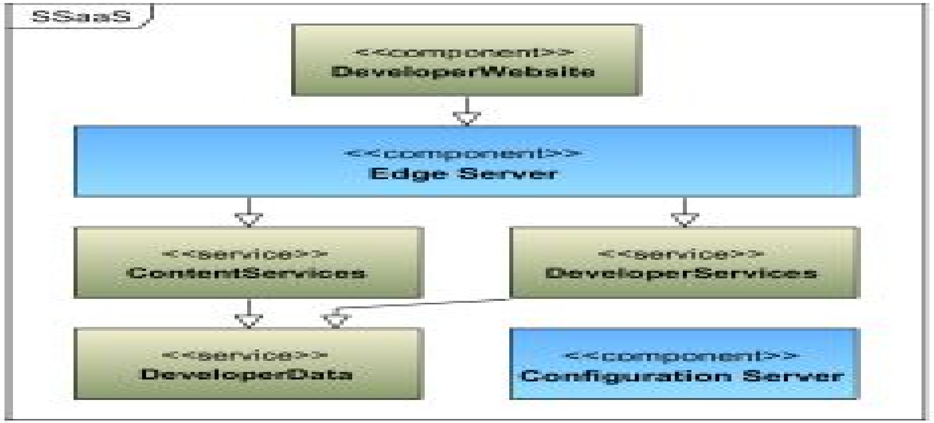
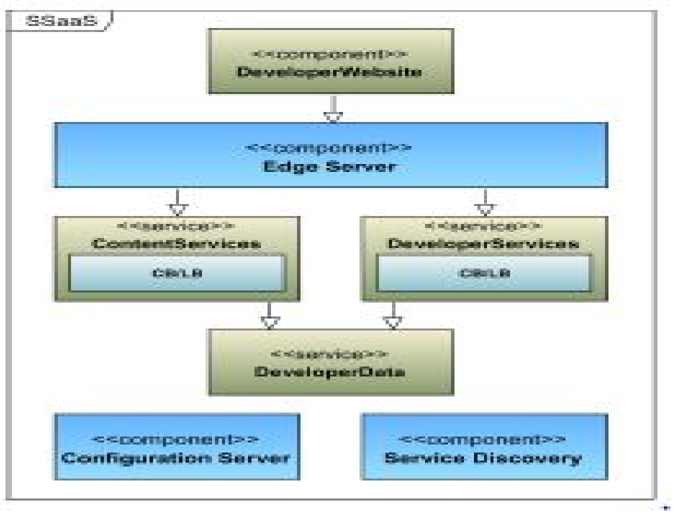


Fig. 5. Introducing Edge Server

Introducing Dynamic Service Collaboration

Addition of Service Discovery, Load Balancer and Circuit Breaker to the system



Introducing Resource Manager

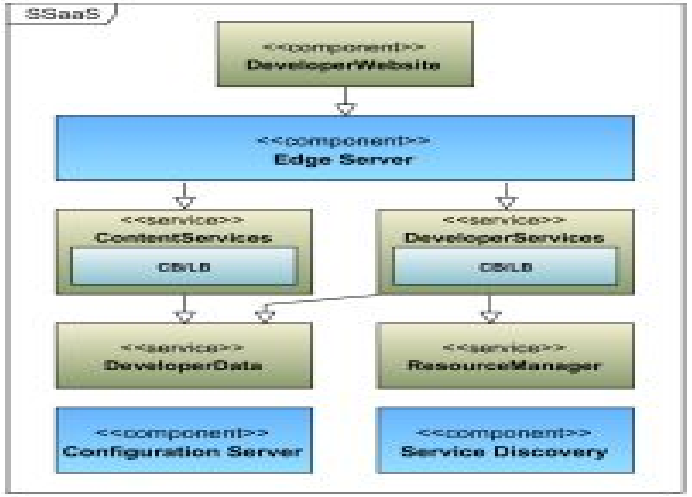


Fig. 7. Introducing Resource Manager

Introducing Chat Services & Developer Info Services

- **Developer InfoServices-** factoring out developer related entities (e.g., Developer) from Developer Data.

- **Chat Services** for persisting chat service instances metadata and handling chat service instance creations.

Clusterization

containerization-low overhead.

Increase efficiency by introducing lightweight operating systems, like Core-OS 15 and Project Atomic

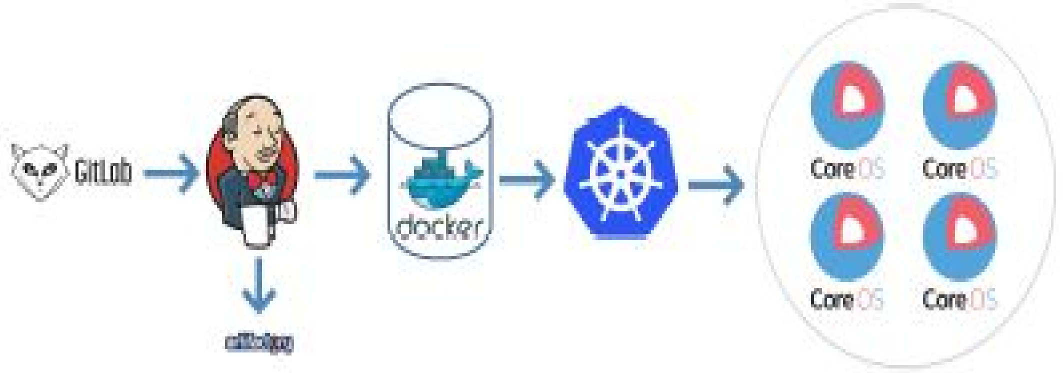
Google Kubernetes 16,has a good integration with the CoreOS, is a tool for easy deployments of containers on a cluster.

Using Kubernetes, a container can be easily fetched from a private repository and deployed to a cluster with different policies.

Final Delivery Pipeline

We set up a cluster of CoreOS instances with Kubernetes agents installed

We deployed our services on this cluster instead of a single server.



Author’s Evaluation and Challenges Faced

- Deployment in the development environment is difficult
- Service contracts are double important
- Distributed system development needs skilled developers
- Creating service development templates is important

Conclusions

- This paper explained the experiences which the author faced during the migration of an on-premise application to the microservices architectural style.
- This paper helped us understand the architecture of our system before and after the migration
- Steps that were followed during this migration process.
- Importance of Continuous Delivery in the process of adopting microservices

Critique: Strengths

- Services are loosely coupled and more modular
- Improves Scalability and Flexibility in a efficient manner
  - Services can be scaled independently based on heavy load instead of scaling the entirety of a monolithic app.
- Fault Isolation
- Freedom of Technology Stack
- Polyglot Programming/ Persistence
  - Leverage mix of programming language /frameworks to take advantage based on business requirement

Critique: Strengths

- Containerization helps in lower overheads than the virtualization and in isolation
- Decentralized data governance
- Automated Deployment with the help of Continuous Delivery pipeline

Critique: Challenges

- Refactoring the design of a system before migration
- Operational Management : Deployment in the development environment is difficult
- Service Versioning is not a recommended solution.
    - Tolerant Reader : Service Consumer
    - Consumer Driven Contracts (Pact.io) : Service Developer
- Knowledge of Distributed System Development
- Security and Firewall

Critique: Evaluation

- IEEE Journal 2016
- Migration in Incremental steps.
  - Re-architecting the current system
  - Introducing new supporting services
  - Enabling Continuous Delivery in the system
- Importance of Continuous Delivery Integration :on-demand software deployment
- Technology Stack
- Experience Report

Gaps

- Worthiness of the migration.
  - Performance Compared to the on-premise
- Cost effectiveness of the migration

Future Work

- DevOps Pattern for on-premise to Cloud migration
  - Reusable
  - Generic

Thank you