IBM Cloud Forum 2020

16 & 17 July 2020

Application Modernization
- 4 types of approach with practical use cases

–

Heesoo Kim
Associate Partner

클라우드로 향하는 길은 여러분을 리더로 이끌 것입니다.

Market Challenge

외부고객

더나은 서비스에 대한 기대

경쟁자

Cloud Native 역량 확보

내부고객

비즈니스 목적에 부합하는 신기술 도입 기대

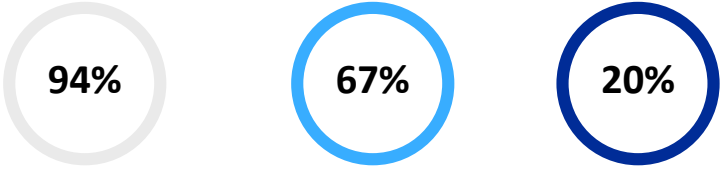
Position for the future

- Innovate
- Speed to Market
- Discover
- Scale

Enable your agility and readiness

- Efficiency
- Risk and Security
- Resiliency
- Lower Costs

A Real World Look at Multicloud



Multiple cloud 사용

1개 이상의 Public cloud 사용

Cloud로 워크로드 전환

Movement between clouds

Consistency of management

Connectivity between clouds

가속화 된 클라우드 변환을 실현하는 데 중요한 요소

Common challenges faced in the journey to cloud

"Am I overpaying for my existing infrastructure investments and how can I improve returns?"

Optimize IT to accelerate digital transformation.

"My business is not agile enough because I have apps my technical team says I can't change quickly."

Modernize applications to increase agility.

"How can I think differently and accelerate the speed of innovation at my company?"

Build cloud-native applications to accelerate innovation.

"Cloud management and security is complicated with multiple cloud applications, providers and other IT environments. How can I get better visibility and control to realize outcomes from my cloud strategy?"

Manage IT to orchestrate and simplify.

Hybrid

Work across public, private and traditional environments.

Multicloud

Adopt virtually any and all clouds that are right for your business

Open

Build once, deploy anywhere with portability

Secured

Realize greater reliability and continuous updates to match dynamic environments

Managed

Gain consistent service-level support across cloud environments

Hybrid & Multi-Cloud Journey 지원 - Red Hat & IBM

OpenShift offers one uniform platform across public and private clouds for full portability, standardization and adoption ease

Linux

Container

Kubernetes

Why Red Hat OpenShift?

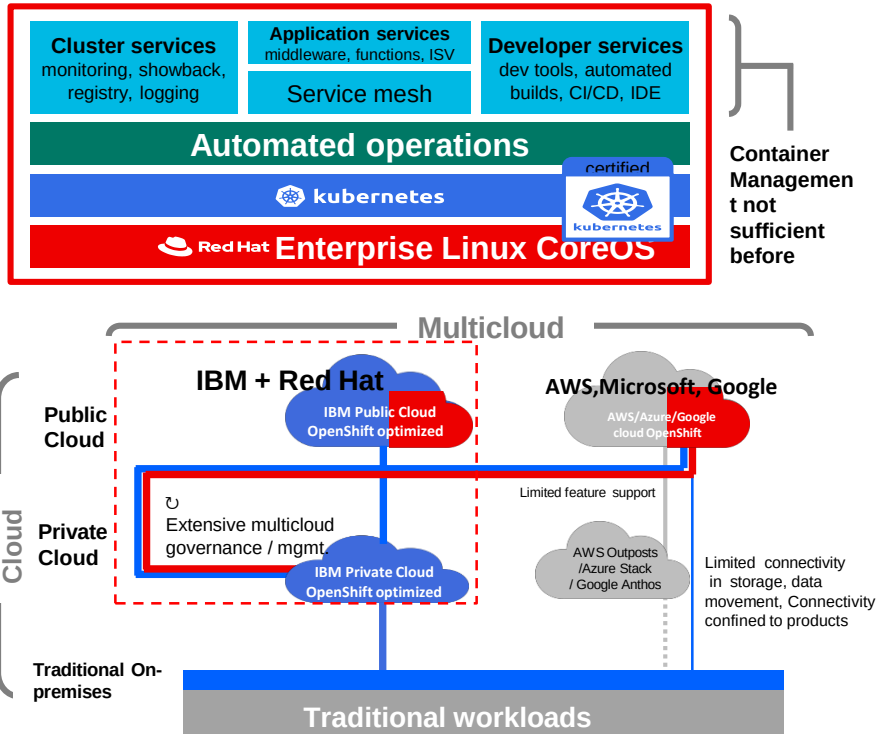
Fully integrated and automated Architecture

Seamless Kubernetes deployment on any environment

Fully automated installation, from cloud infrastructure to OS to application services

One click platform and application updates

Auto-scaling of cloud resources



IBM 이 제공하는 6주간의 enablement 프로그램
: Co-creates, co-executes, and co-operates

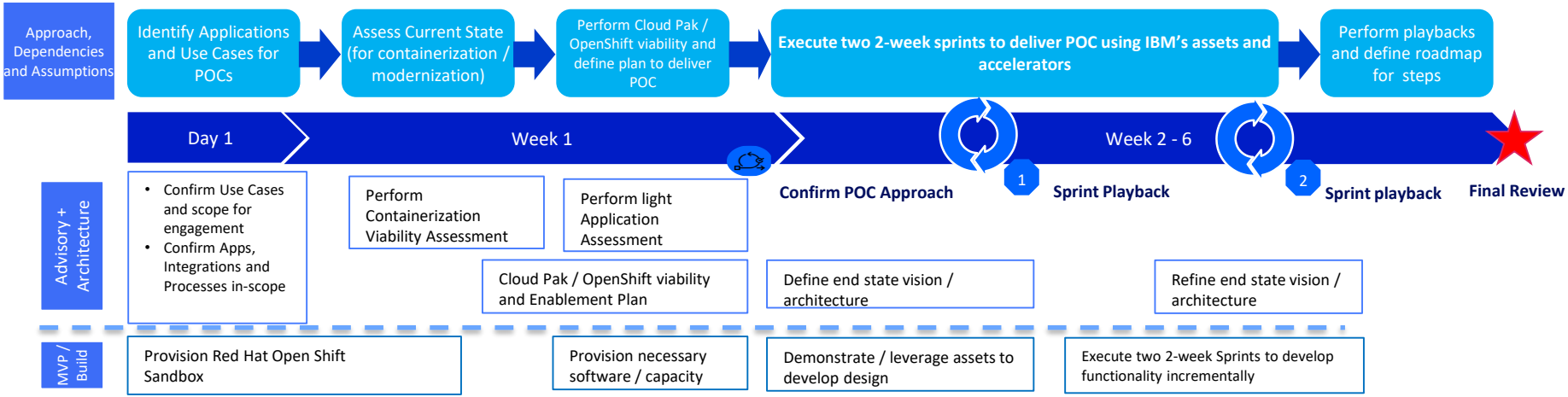
IBM Services & Red Hat OpenShift Benefits

Cost Savings
Up to 30% reduced operating expenses

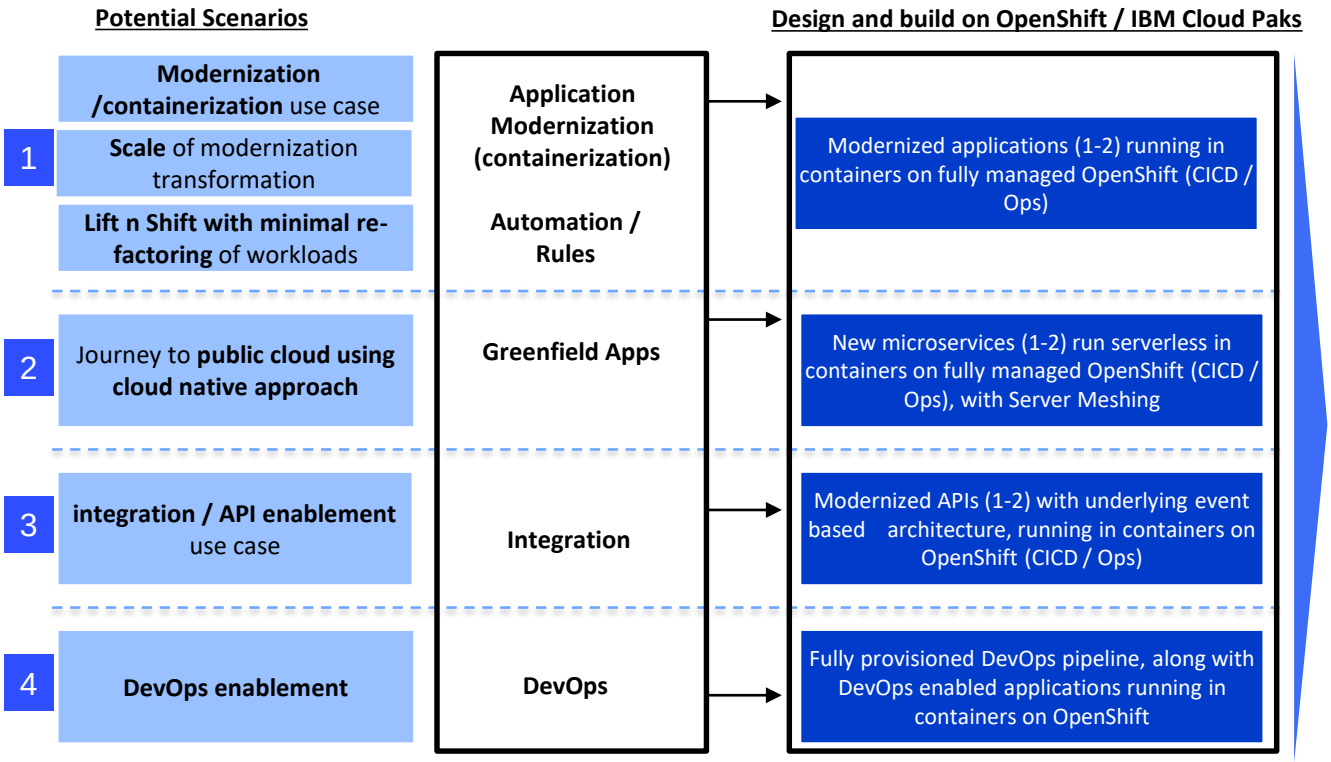
Faster development
Up to 50% faster time to market than traditional methods

Accelerated Hybrid Cloud Journey
acceleration of migration and modernization

Culture promoting agility and continuous learning



Use-Case Driven Delivery Model – 4 Types



Deliverables Summary

- Light Application Assessment
- Working POC
- Cloud Value Model
- Next steps to execute the end state vision

Desired Outcomes

- Demonstrate benefits of Red Hat OpenShift as the container platform for Hybrid Multi-cloud
- Prove value of the IBM Services assets to accelerate, simplify and standardize cloud and application modernization

Application Modernization - POC Case
: 채널 어플리케이션을 컨테이너 기반의 마이크로서비스로 전환 개발

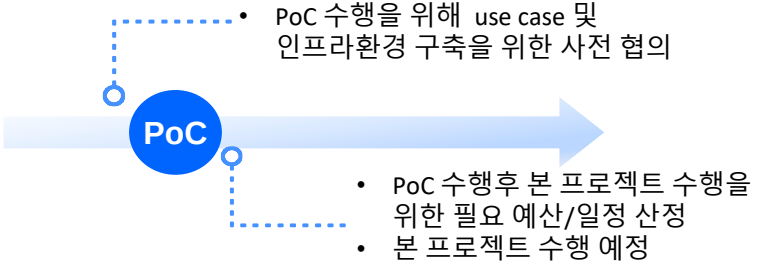
–

PoC Case 1 : 은행고객사

채널 어플리케이션을 마이크로서비스로 전환한 케이스

Application Background

- **PoC 수행기간** : 2020.04.20 ~ 2020.05.15
- **Why PoC** : 은행 고객사의 비대면 채널 시스템을 Cloud Native 환경으로의 전환을 적용하여 은행의 Cloud Transformation 전략적 방향성에 맞추기 위해 진행
- 구현 범위는 2주간 구현 가능한 범위로 MSA 식별하여 구현



PoC (Proof of Concept) 범위

PoC 는 예금상품 조회, 예금계좌 조회, 환율조회 3가지의 Use Case를 갖고 마이크로서비스를 구현함

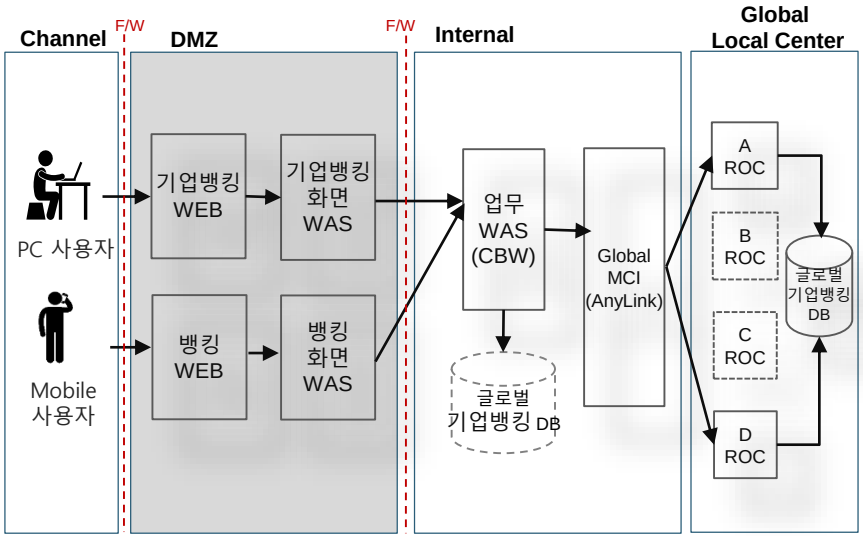
3 Use Case

로그인 | 예금상품 정보조회 | 예금계좌 조회 | 환율 조회

- **마이크로서비스 개발** – 클라우드에 최적화된 개발 프레임워크 사용, 로그인 서비스 구현, 마이크로서비스 구현, 새로운 디자인 적용
- **컨테이너화 적용** – 마이크로 서비스 기반의 아키텍처구성, 서비스 매쉬 적용, Scale-Out 구성
- **CI/CD 적용** – 고객사 DevOps CI/CD적용

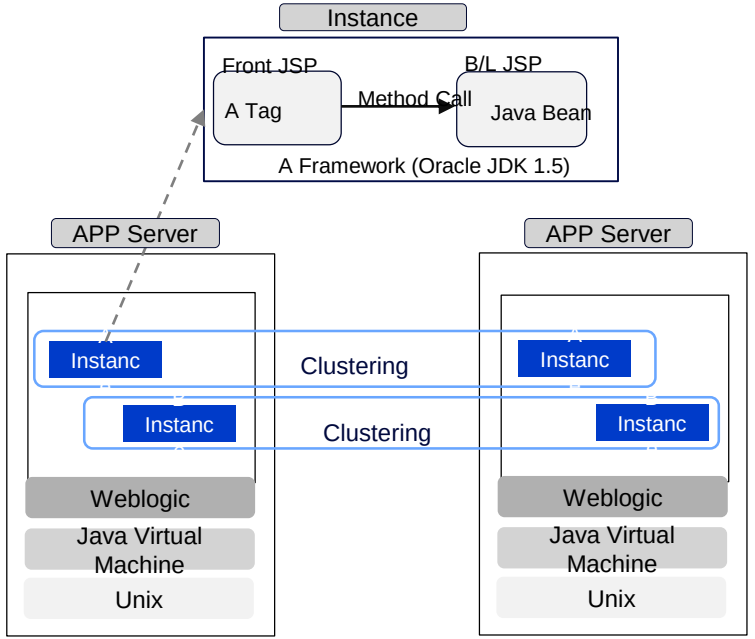
PoC 수행 범위 > As-is 아키텍처 & SW 구조

As-is 아키텍처



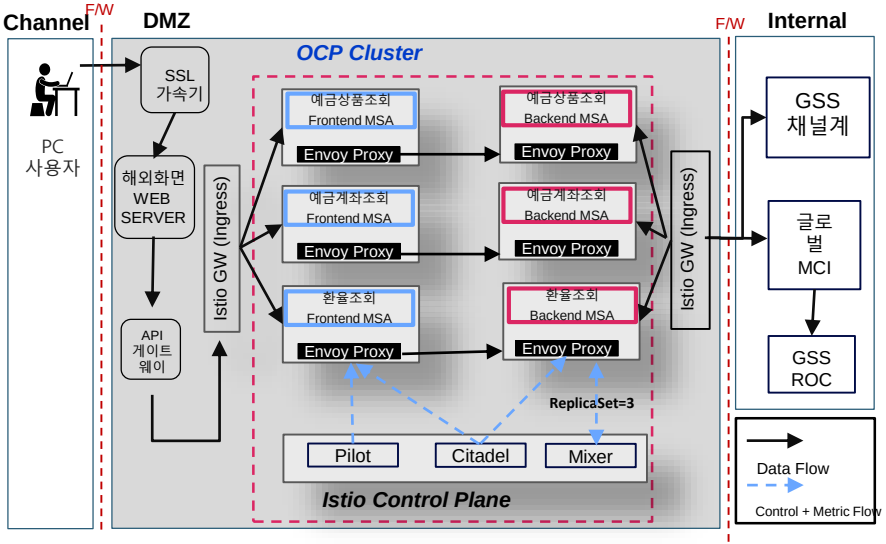
- 별도의 분리된 시스템 구성없이 국내 기업뱅킹/뱅킹 시스템과 통합하여 사용하고 있음

As-is SW 구조



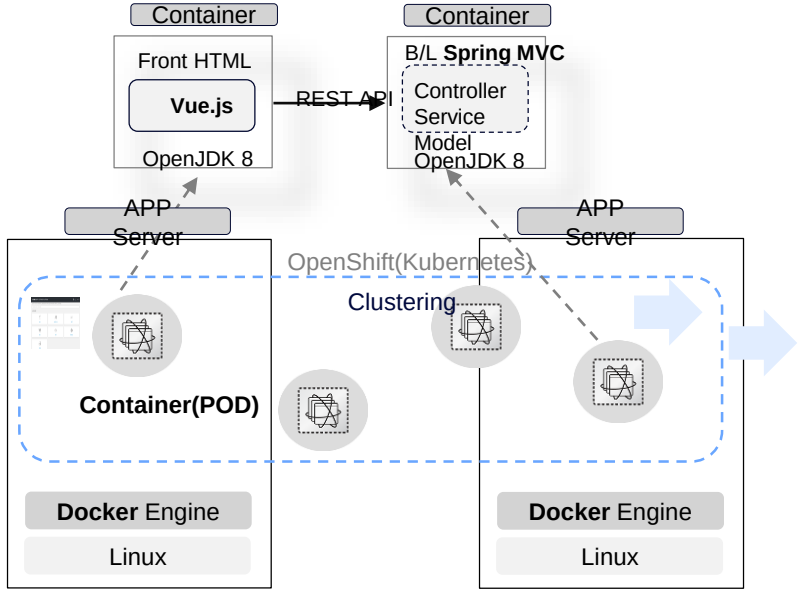
PoC 수행 범위 > To-be 아키텍처 & SW 구조

To-be 아키텍처



- IaaS 와PaaS가 같이 운영되는 Hybrid 방식
- Functional 기능 외에도 Non-Functional 기능 요건 면밀히 검토 및 설계 필요

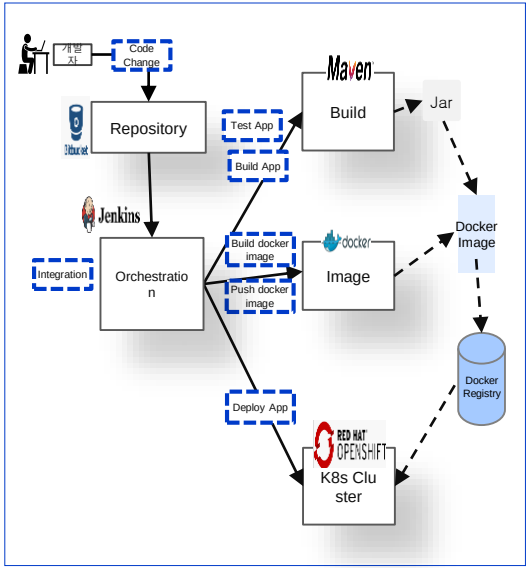
To-be SW 구조



- 개발 프레임워크 구조를 A Framework 에서 Vue.js + Spring Boot 기반으로 변경하여 개발을 수행함

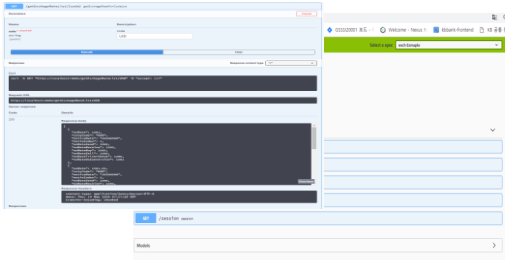
PoC 수행 결과 > CI/CD Toolchain및 API 문서화 및 테스트 방법

Jenkins 기반의 Toolchain 구성



별도의 분리된 시스템 구성없이 국내 기업뱅킹/뱅킹 시스템과 통합하여 사용하고 있음

API 문서화 및 테스트 방법



- API 문서를 **코드 내에서 주석기능을 이용해** 작성하고, 웹페이지로 확인하는 기능을 제공하는 프레임워크
- 소스코드 빌드시 API 문서 자동 생성되기 때문에 **별도의 문서 작업없이 개발에 집중할 수 있음**
- API 테스트를 위해 데이터를 입력하고 버튼을 클릭하는 것으로 **실제 통신을 보내고 응답을** 받을 수 있는 기능 제공
- **개발툴에 Plugin 방식으로** 지원하기 때문에 사용하기가 쉬움

Swagger를 이용하여 API 문서화 및 테스트를 수행함

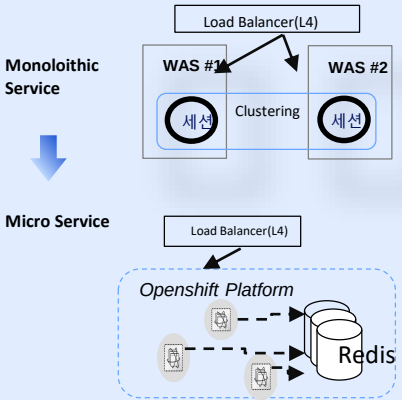
PoC 개발환경

대분류	소분류	구성항목
개발 환경	개발환경 전략	• Open Source 기반의 S/W 사용
	개발 프레임워크	• **Spring Boot 2.1.6 사용**
	UI 개발 프레임워크	• **Vue.js**
	개발도구	• **STS(Springboot Tool Suite) 4.6** *이클립스 기반
	개발소스 저장소	• **Git 기반의 Bitbucket 연동**
	빌드환경	• **Maven 4.6 사용**
	프로젝트 구성	• 총 6개의 멀티 프로젝트 구성
	Library 및 Docker 이미지 저장소	• **Nexus 구성**
	로그정책	• **Log4J**
개발 방식	**설정파일 파일관리**	• **OCP Configmaps 활용**
	설정파일 관리정책	• IP 및 Port 사용불가, 하드코딩 금지
	프로젝트 관리도구	• **JIRA, Confluence 활용**
	Service Mesh 구현	• **어플리케이션 Layer에서 구현없이 인프라에 구성된 Istio 사용**
	MCI/EAI 연동	• **JSON 형태의 Mockup(echo file) 활용** *Global MCI 개발진행중
개발 운영 환경	**컨테이너 이미지**	• **Tomcat, Redis, MySQL**
	컨테이너별 차등옵션 여부	• 컨테이너별 차별옵션 불가
	형상관리 및 배포관리	• **CI/CD Devops 체계 활용**
	배포형상 확인	• **Jenkins 활용**
	Istio 설정	• Gateway, Circuit Breaker 구성

PoC 수행 결과 > Scenario 요약

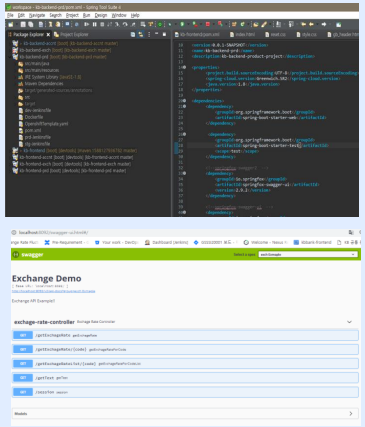
1 마이크로서비스 동작방식 및 세션관리

마이크로 서비스 동작방식을 설명하고 마이크로 서비스 방식에서의 세션관리 방안에 대해 검증함



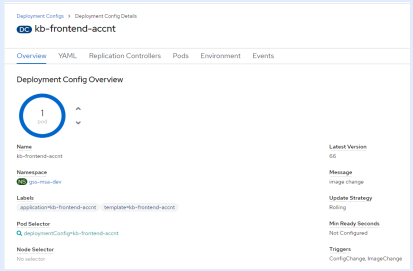
2 마이크로서비스 개발환경구조 및 배포방식

POC에서 수행한 어플리케이션 구조와 배포방식을 설명하고 마이크로 서비스에 변경이 발생해도 다른 마이크로서비스에 영향을 미치지 않는다는 것을 검증함



3 컨테이너 기반 서비스의 Auto-Scaling

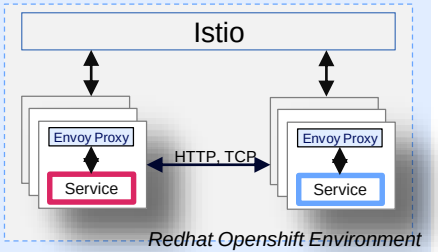
비정상적인 트래픽 증가에 대응하기 위해 리소스 사용률, 사용자 지정 메트릭등을 기반으로 자동으로 Pod의 복제본이 증가하는 Auto Scaling 기능을 검증함



4 Istio 기반의 서비스 메시 (Service Mesh)

컨테이너 기반 서비스에서 비정상 동작으로 인해 오류가 발생할 경우 어떠한 방식으로 장애가 전파되지 않도록 하는지 확인하고 Istio에서 제공하는 기능들이 어떠한 것들이 있는지 소개

Istio?

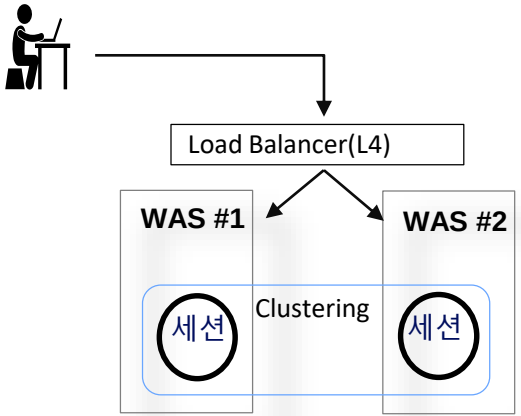


서비스와 서비스 사이에 발생하는 행동들에 대해 네트워크 구간에서 Authentication, Tracing, Logging, Monitoring 할 수 있는 기술

PoC 수행 결과 > 1 마이크로서비스 동작방식 및 세션관리 Scenario

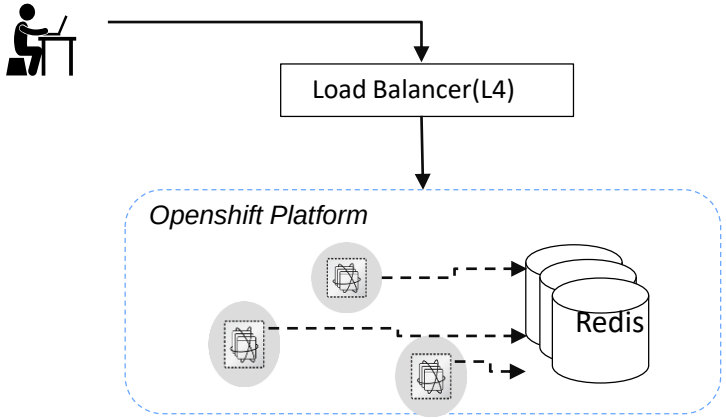
마이크로 서비스 동작방식을 설명하고 마이크로 서비스 방식에서의 세션관리 방안에 대해 검증함

Monolithic Service



1. 세션이 공유되어 있지않다면 WAS #1에 접속한 사용자가 WAS #2에 접속할 경우 로그인이 되어있지 않다고 표시됨
2. 이를 해결하기 위해서 WAS 환경설정에 IP기반의 설정을 통해 서버간 Clustering을 구성

Micro Service

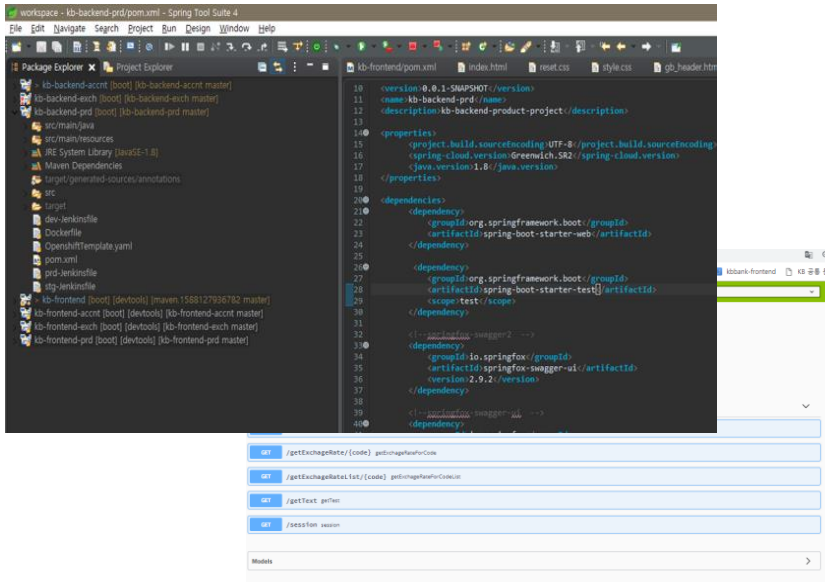


1. 컨테이너 서비스의 경우 IP가 아닌 도메인 기반으로 동작이기 때문에 컨테이너 상에 별도의 설정을 할 수 없음
2. 컨테이너가 동적으로 Scale-Out되더라도 별도의 설정없이 Redis의 저장소에 있는 세션정보를 공유받을 수 있음
3. Redis간 클러스터링 지원을 통해 Redis 자체의 부하를 분산시킬 수 있음

PoC 수행 결과 > 2 마이크로서비스 개발환경구조 및 배포방식 Scenario

POC에서 수행한 어플리케이션 구조와 배포방식을 설명하고 마이크로 서비스에 변경이 발생해도 다른 마이크로서비스에 영향을 미치지 않는다는 것을 검증함

백그라운드 설명



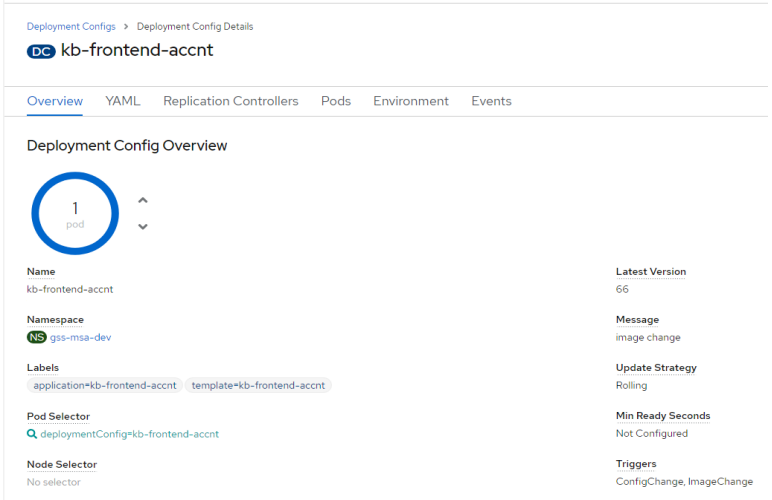
시연 시나리오

1. 어플리케이션 구조 및 배포방식 설명
2. 실제 개발툴을 띄워 환율조회 프로젝트에서 메시지를 변경
3. 수정된 정보를 Commit한 후 Jenkins에서 환율조회 Pipeline 빌드
4. Openshift상에 포드가 Build되는 것을 확인한 후 환율조회 서비스 클릭하여 메시지가 변경되었는지 확인
5. Kiali를 통해 서비스 호출 흐름 설명
6. Swagger 테스트 도구 설명

PoC 수행 결과 > 3 컨테이너 기반 서비스의 Auto-Scaling Scenario

비정상적인 트래픽 증가에 대응하기 위해 리소스 사용률, 사용자 지정 메트릭등을 기반으로 자동으로 Pod의 복제본이 증가하는 Auto Scaling 기능을 검증함

백그라운드 설명



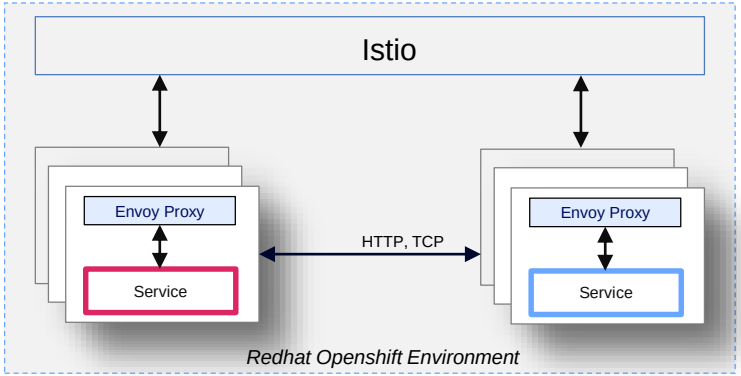
시연 시나리오

1. POD가 1개일때의 정보 확인 (Deploy Config – POD 확인, Istio-Grafana에서 CPU 확인)
2. Target CPU 사용률을 30%로 설정
3. Jmeter를 활용하여 CPU를 80%까지 증가
4. POD가 3개일때의 정보 확인 (Deploy Config – POD 확인, Istio-Grafana에서 CPU 확인)

PoC 수행 결과 > 4 stio 기반의 서비스 메시(Service Mesh) Scenario

컨테이너 기반 서비스에서 비정상 동작으로 인해 오류가 발생할 경우 어떠한 방식으로 장애가 전파되지 않도록 하는지 확인하고 Istio에서 제공하는 기능들이 어떠한 것들이 있는지 소개

백그라운드 설명



주요기능

서비스와 서비스 사이에 발생하는 행동들에 대해 네트워크 구간에서 Authentication, Tracing, Logging, Monitoring 할 수 있는 기술

- Retry, Timout, **서킷 브레이커(Circuit Breaker) 등의 기능 제공**
- 트래픽 통제, 컨텐츠 기반의 트패픽 분할
- **트래픽 모니터링 및 시각화 지원**
- 보안(Security) 및 인증(Authentication) 지원

시연 시나리오

1. Service 요청에 대한 Connection pool 정의
2. 최대 활성 연결 갯수와 최대 요청 대기수 지정
3. 허용된 최대 요청 수 보다 많은 요청을 하게되면 지정된 임계값만큼의 대기요청 수행
4. 이 임계점이 넘어가는 추가적인 요청은 Circuit Breaker 발동하는지 확인
5. Istio 와 연계된 Grafana, Kiali 서비스 소개

To-be 고려사항 (Containerization, Microservice 설계 및 구현)

1	인프라 아키텍처	• 서비스가 IaaS 에서 PaaS로 전환에 따른 DR 구성 등 인프라 아키텍처 검토
2	솔루션 호환성	• 솔루션 컨테이너화 가능여부 및 라이센스 확인
3	보안	• 서버 보안, 컨테이너 보안 가이드 • 취약성 점검 수행 방법
4	로그 통합	• 로그 보안, 로그 추적방안 검토
5	모니터링	• 예방적 장애대응에 대한 기준 마련 • 장애 대응 모니터링, 알람 구성 방안
6	DevOps 체계	• 배포 및 릴리즈 작업 자동화 • SR 처리 프로세스


IBM