

Service Mesh & CI/CD with AWS DevOps Pipeline

10th Jan 2022, ver2.1



Copyright © 2021. uEngine-solutions All rights reserved.

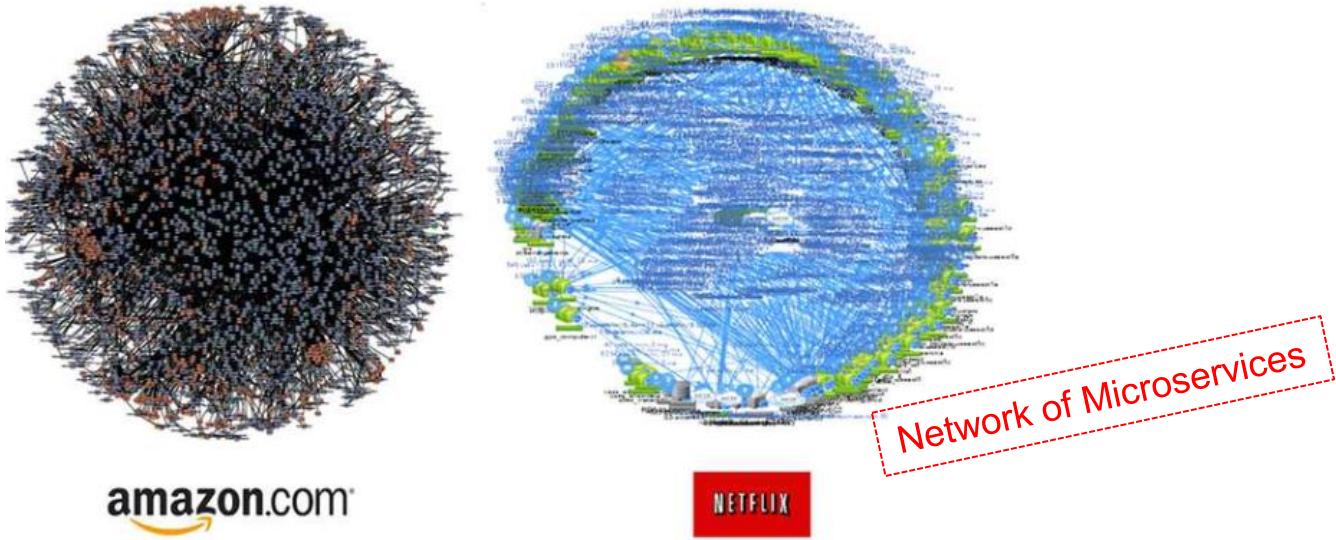
 MSA School

 uEngine

Table of content

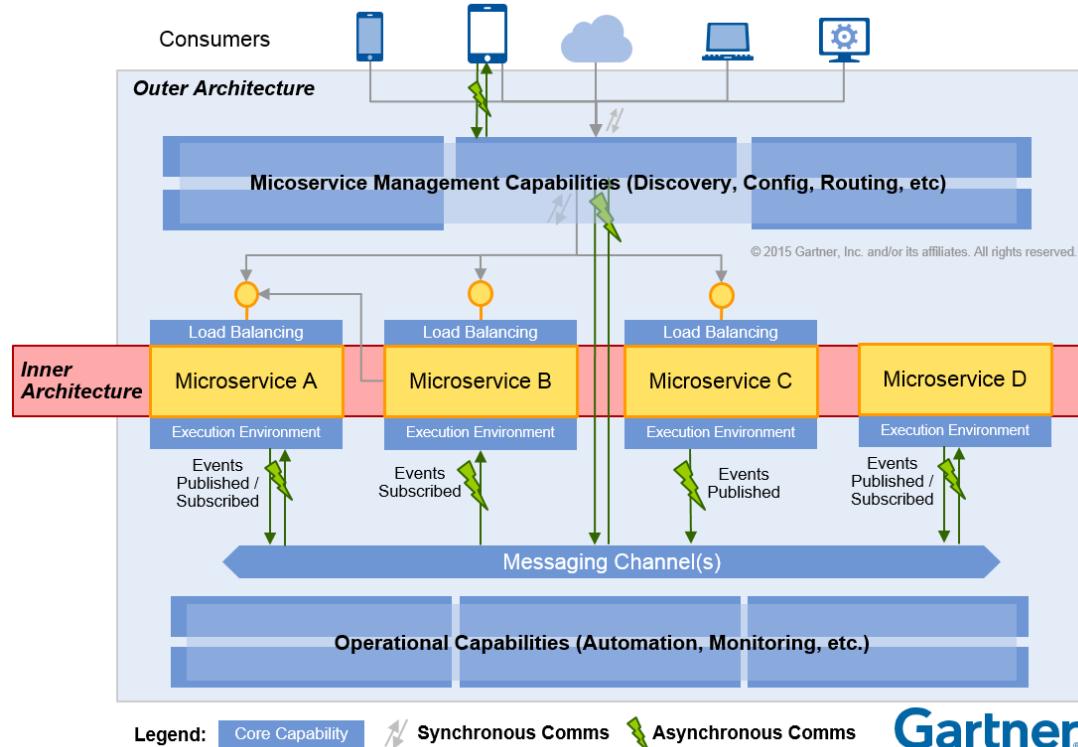
1. Service Mesh: Istio ✓
2. DevOps Process and Container-based Pipeline
3. Version Control(SCM) & Build Automation Tools
4. AWS CodeStar
5. AWS CodeBuild

Service Mesh- Service to Service Communicator



- Service Mesh는 마이크로서비스 간의 통신(네트워크)을 담당하는 요소
- 통신 및 네트워크 기능을 비즈니스 로직과 분리한 네트워크 통신 인프라
- 서비스간 통신을 위해서는 Service Discovery, Fault Tolerance, Traffic Routing, Security 등의 기능 필요

Service Mesh in Microservice Architecture



Inner Architecture:

개별 마이크로서비스 영역으로 가능한 간결, 명확하고 단순하게 구현하는데 중점

Outer Architecture:

- 마이크로서비스 외부를 아우르는 영역으로 트래픽 통제 및 서비스간 어떻게 커뮤니케이션 하는가 등, 클라우드 담당자가 공히 겪게 되는 복잡성들 (e.g. 서비스 발견, 탄력성, 장애 내선 등)을 네트워크 레벨에서 처리

(Service Mesh 영역)

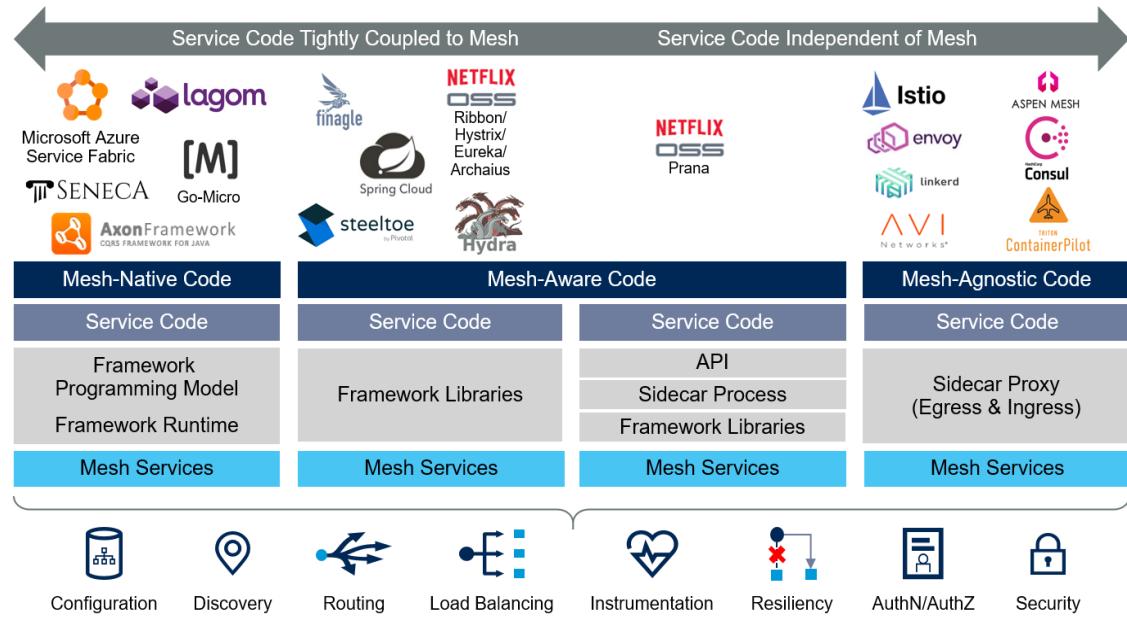
API Gateway vs. Service Mesh

- API Gateway는 마이크로서비스 그룹의 외부 경계에 위치하여 역할을 수행하지만, Service Mesh는 경계 내부에서 그 역할을 수행

	API Gateway	Service Mesh
라우팅 주체	<ul style="list-style-type: none">서버	<ul style="list-style-type: none">요청하는 서비스
라우팅 구성요소	<ul style="list-style-type: none">별도의 네트워크를 도입하는 독립적인 API Gateway 구성요소	<ul style="list-style-type: none">Service 내 SideCar로 Local network 스택의 일부가 됨
로드 밸런싱	<ul style="list-style-type: none">단일 엔드포인트를 제공하고, API Gateway 내 로드밸런싱을 담당하는 구성요소에 요청을 redirection하여 해당 구성요소가 처리함	<ul style="list-style-type: none">Service Registry에서 서비스 목록을 수신함. Sidecar에서 로드밸런싱 알고리즘을 통해 수행함
네트워크	<ul style="list-style-type: none">외부 인터넷과 내부 서비스 네트워크 사이에 위치함	<ul style="list-style-type: none">내부 서비스 네트워크 사이에 위치하며, 응용프로그램의 네트워크 경계 내에서만 통신을 가능하게 함
트레이싱	<ul style="list-style-type: none">API에 대한 사용자 및 공급자에 대한 모든 호출에 대해 수집되고 분석됨	<ul style="list-style-type: none">Mesh 내 모든 마이크로서비스 구성요소에 대해 분석 가능

Service Mesh Types

A Spectrum of Service Mesh Technologies Based on Code Coupling



< 서비스 메시의 종류 >

• 코드 기반

프레임워크 기반 프로그래밍 모델로 서비스 메시를 구현하는데 특화된 코드가 필요한 유형

• 라이브러리 기반

라이브러리로 구현되어 API호출을 통해 Service Mesh에 결합되는 유형

• Proxy 기반

Istio/ Envoy, Consul, Linkerd 등 Sidecar Proxy를 통해 마이크로서비스에 주입하는 유형

Service Mesh Comparasion

Aspect	Sidecar Proxy (Istio/Envoy)	Library (Spring Cloud & Netflix OSS)
Application Language	<ul style="list-style-type: none">Polyglot	<ul style="list-style-type: none">Java
Architecture	<ul style="list-style-type: none">사이트카 패턴 (uses Envoy as proxy)	<ul style="list-style-type: none">애플리케이션에 라이브러리 추가
Implementation Level	<ul style="list-style-type: none">Platform (using YAML)	<ul style="list-style-type: none">Application (using Code)
Service Discovery	<ul style="list-style-type: none">Implicit (자동 등록)	<ul style="list-style-type: none">Explicit (명시적 선언)
Circuit Breaker	<ul style="list-style-type: none">YAML Configuration	<ul style="list-style-type: none">Coding/ Annotation
Tracing	<ul style="list-style-type: none">Envoy Proxy들을 통한 호출 추적	<ul style="list-style-type: none">추적 데이터를 명시적으로 로깅
Security	<ul style="list-style-type: none">Service to ServiceMutual TLS through Envoy	<ul style="list-style-type: none">Application Specific

Istio Service Mesh key features

- Improved Routing & Deployment Strategy
- Improved Smart Resilience
- Improved Security
- Improved Observability

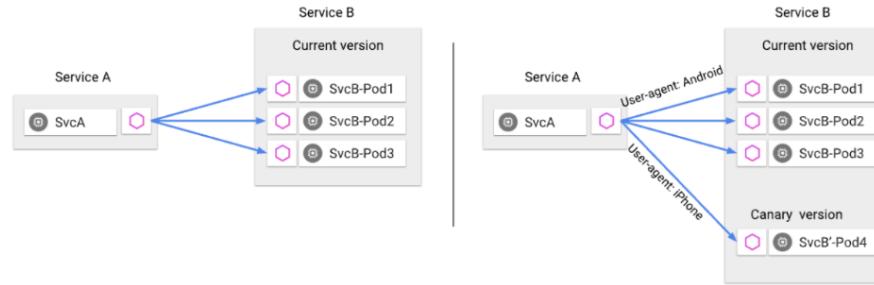


- ✓ Service Discovery
- ✓ Load balancing
- ✓ Dynamic Request Routing
- ✓ Circuit Breaking
- ✓ 암호화 (TLS)
- ✓ 보안
- ✓ Health check, Retry and Timeout
- ✓ Metric 수집



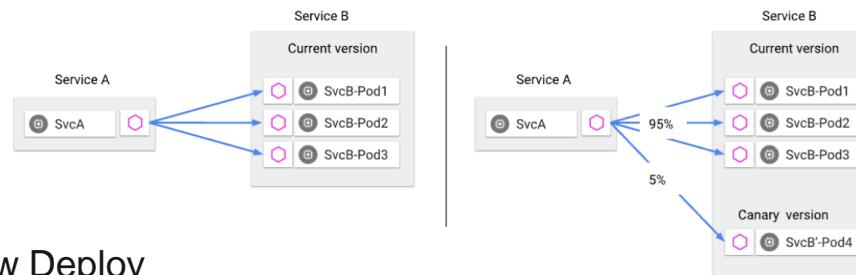
1. Improved Routing & Deployment Strategy

- Traffic Routing Controls



- Canary Deploy

- 특정 유저의 신상, 지역, 권한, 접근 단말에 따른 다른 버전의 노출



- A/B Testing, Shadow Deploy

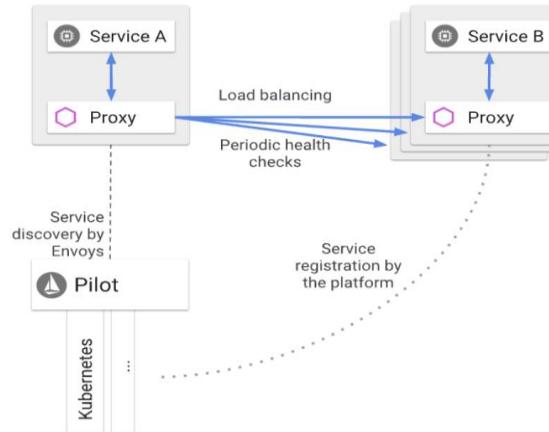
- 신규 버전의 오류 노출 없는 실질적 테스트

2. Improved Resilience

- Retry
 - 서비스간 호출의 실패에 대한 재시도
- Circuit Breaking / Rate Limiting
 - 인스턴스의 보호
 - 전체 서비스 장애 차단
- Pool Ejection
 - 죽은 인스턴스의 제외
- Health Check & Service Discovery
 - 여러 인스턴스에 대한 로드 밸런싱
 - 서비스 앤드 포인트 관리



Circuit Breaking + Pool Ejection + Retry
= High Resilience

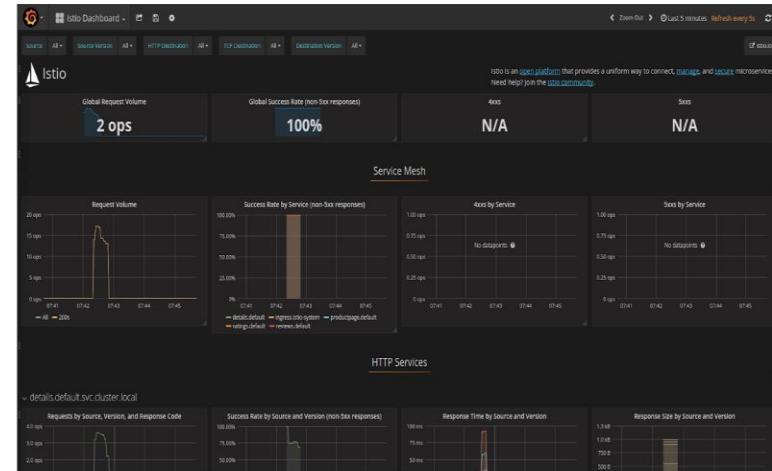
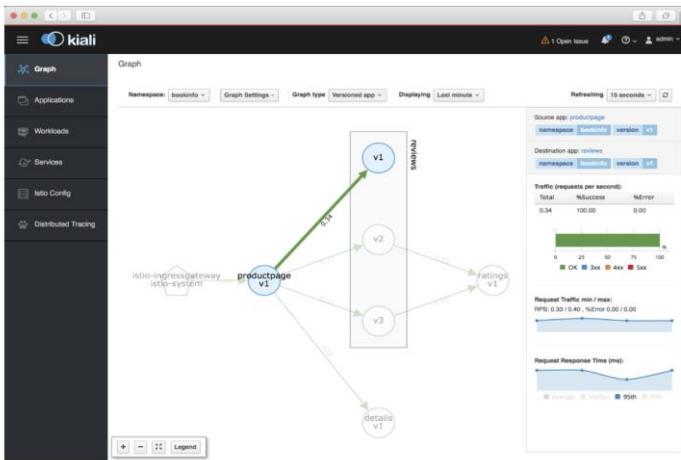


3. Improved Security

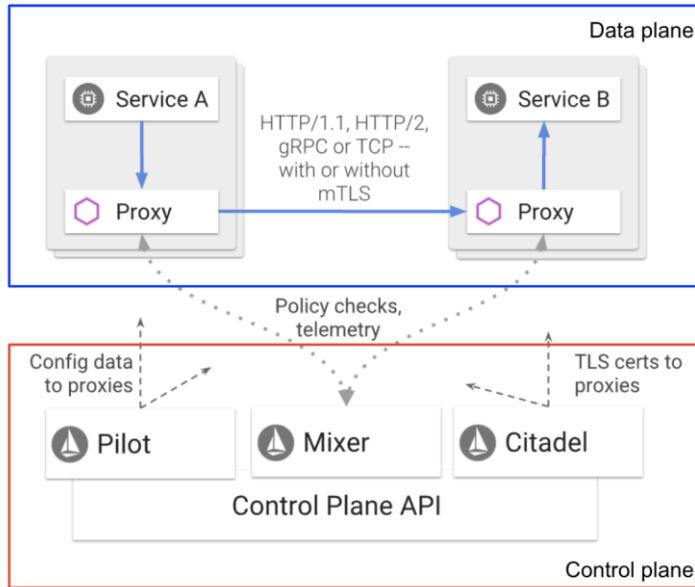
- TLS based Inter Microservices Communication
 - Envoy로 통신하는 모든 트래픽을 자동으로 TLS를 이용해 암호화
- Service Authentication & Authorization
 - JWT Token을 이용한 서비스 접근 Client 인증
 - 역할기반(RBAC) 권한 인증 지원
- Whitelist and Blacklist

4. Improved Observability

- Distributed Tracing and Measure
 - 서비스간 호출 내용 기록
 - Istio 설치 시, Kiali, Jaeger가 default로 설치되어 각각 Monitoring 및 Tracing 지원



Istio Service Mesh architecture



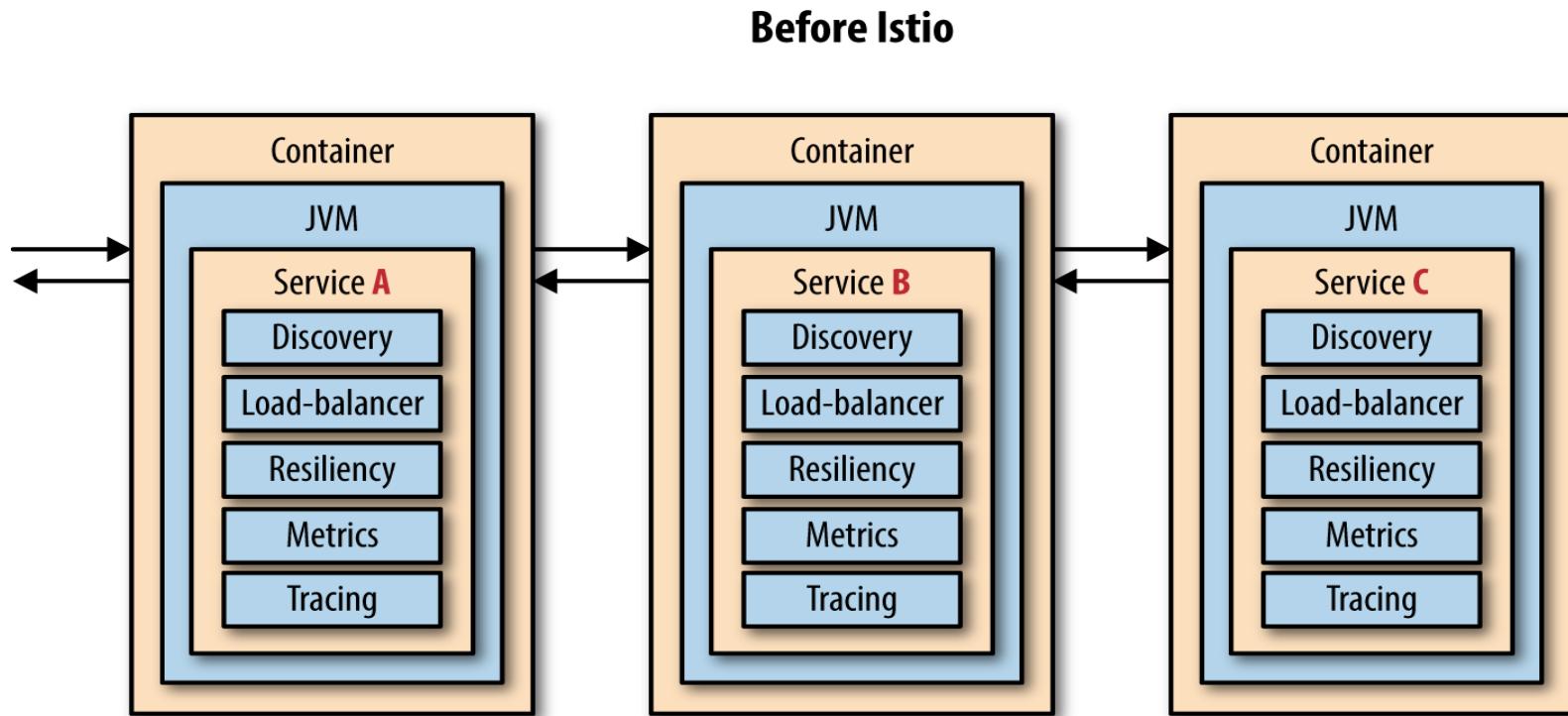
- **Data Plane(데이터 플래인)**

- 서비스 옆에 사이드카(Envoy)를 붙여, 서비스로 들어오고 나가는 트래픽을 Envoy를 통해 통제
- 유입/유출 트래픽 통제 : Ingress/Egress Controller

- **Control Plane(컨트롤 플래인)**

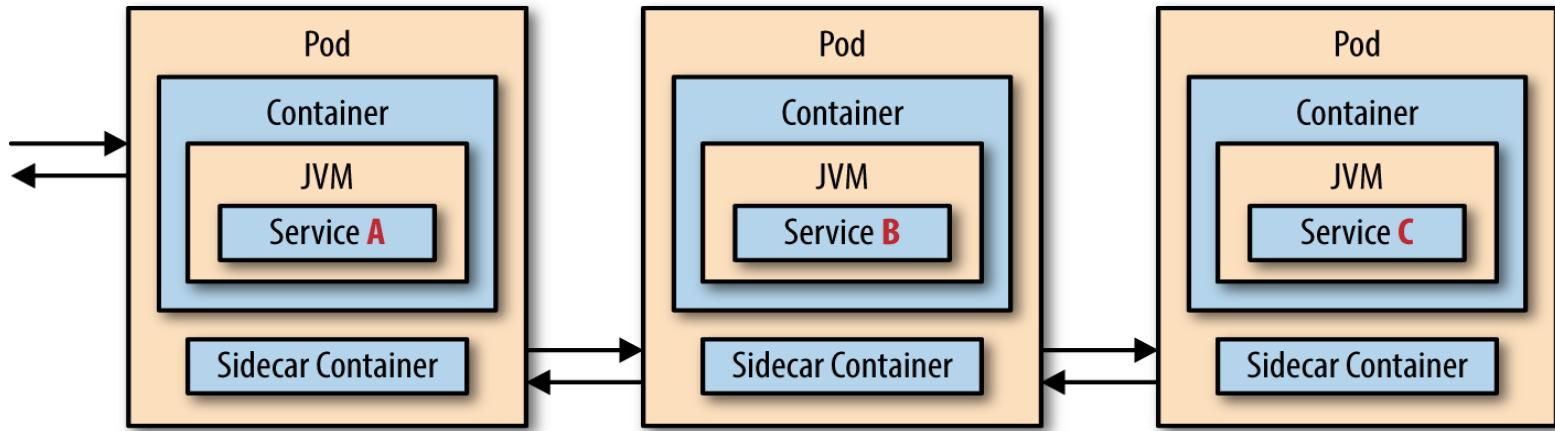
- **Pilot(파일럿)** : Envoy에 대한 설정 관리 및 서비스 디스커버리 기능 제공
- **Mixer(믹서)** : 액세스 컨트롤 및 다양한 모니터링 지표 수집
- **Citadel(시터덜)** : 보안관련 기능 담당 모듈로 사용자 인증/ 인가 및 TLS 통신을 위한 인증서 관리

Before Istio : Spring Cloud + Netflix



After Istio

Envoy



좋은점 :

1. L7 레이어를 사용, 성능이 높음
2. Code 변경 없이 Cross-cutting 이슈를 다루어 줌
3. Main 서비스의 재배포 없이 Sidecar 를 관리 가능함

Lab. Istio Install

- Istio 설치

- curl -L https://git.io/getLatestIstio | ISTIO_VERSION=1.11.5 sh -
- cd istio-1.11.5
- export PATH=\$PWD/bin:\$PATH
- for i in install/kubernetes/helm/istio-init/files/crd*yaml; do kubectl apply -f \$i; done
- kubectl apply -f install/kubernetes/istio-demo.yaml

- 설치확인

- \$ kubectl get pod -n istio-system



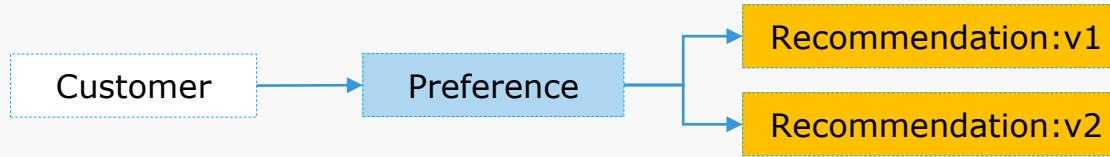
```
apexacme@APEXACME:~/istio-1.4.5$ kubectl get pod -n istio-system
NAME                               READY   STATUS    RESTARTS   AGE
grafana-6bb6bcf99f-78p9q           1/1    Running   0          76s
istio-citadel-597559bf4-lpqxc     1/1    Running   0          75s
istio-egressgateway-5c68d8857-qp14x 0/1    Running   0          76s
istio-galley-769849c5fb-cpvkp      0/1    Running   0          76s
istio-grafana-post-install-1.4.5-frqk4 0/1    Completed  0          78s
istio-ingressgateway-665c7c6b8-k9qhg 0/1    Running   0          76s
istio-pilot-6bd54f544b-65tnf       1/2    Running   1          75s
istio-policy-6b7d856769-xcip2      2/2    Running   2          75s
istio-security-post-install-1.4.5-p9wtc 0/1    Completed  0          78s
istio-sidecar-injector-6d9f967b5-v157x 1/1    Running   0          75s
istio-telemetry-657bbb5677-pg2th    2/2    Running   2          75s
istio-tracing-56c7f85df7-71lkd8    1/1    Running   0          75s
kiali-7b5c8f79d8-dm46s             1/1    Running   0          76s
prometheus-74d8b55f54-r9w7v        1/1    Running   0          75s
apexacme@APEXACME:~/istio-1.4.5$
```

Lab. Istio Tutorial Setup (1/2)

- Git repository에서 Tutorial 리소스 가져오기
 - cd ~
 - mkdir git
 - cd git
 - git clone https://github.com/redhat-developer-demos/istio-tutorial
 - cd istio-tutorial
 - 네임스페이스 생성
 - kubectl create namespace tutorial
 - Customer 애플리케이션 배포
 - kubectl apply -f <(istioctl kube-inject -f customer/kubernetes/Deployment.yml) -n tutorial
 - kubectl create -f customer/kubernetes/Service.yml -n tutorial
 - kubectl create -f customer/kubernetes/Gateway.yml -n tutorial
 - 배포 확인 및 외부접속 설정
 - kubectl -n tutorial edit svc customer
 - (ServiceType : ClusterIP → LoadBalancer로 변경)
 - kubectl -n tutorial get svc
 - 브라우저로 EXTERNAL-IP:8080 접속
- | NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |
|------------------|--------------|-------------|----------------|----------------|------|
| service/customer | LoadBalancer | 10.0.254.95 | 52.231.113.110 | 8080:30926/TCP | 6m4s |

Lab. Istio Tutorial Setup (2/2)

- 서비스 흐름



- 테스트 애플리케이션 배포 (preference, recommendation)
 - kubectl apply -f <(istioctl kube-inject -f preference/kubernetes/Deployment.yml> -n tutorial
 - kubectl create -f preference/kubernetes/Service.yml -n tutorial
 - kubectl apply -f <(istioctl kube-inject -f recommendation/kubernetes/Deployment.yml> -n tutorial
 - kubectl create -f recommendation/kubernetes/Service.yml -n tutorial

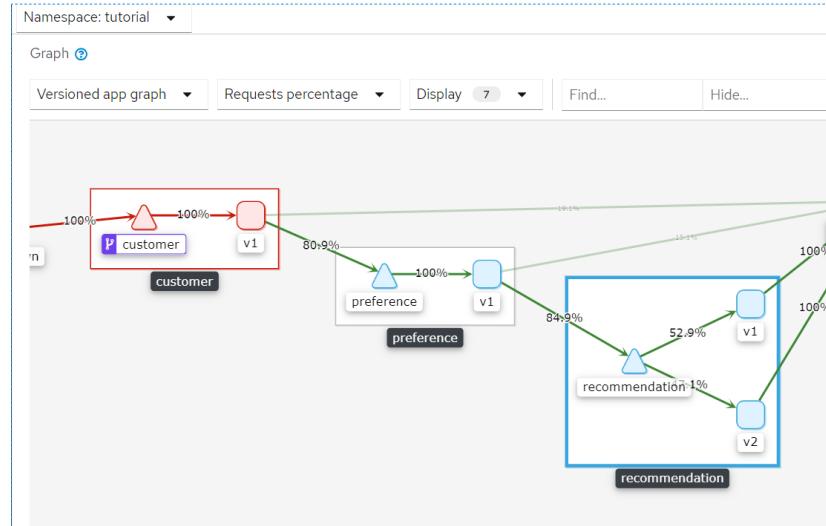
- 모니터링시스템 외부접속 설정

- kubectl edit svc jaeger-query -n istio-system
 - (ServiceType : ClusterIP → LoadBalancer로 변경)
 - kubectl edit svc kiali -n istio-system
 - (ServiceType : ClusterIP → LoadBalancer로 변경)
 - kubectl get svc -n istio-system
 - Jaeger 접속 : EXTERNAL-IP :16686
 - Kiali 접속 : EXTERNAL-IP:20001

jaeger-query	LoadBalancer	10.0.181.81	52.231.113.45	16686:32721/TCP
kiali	LoadBalancer	10.0.217.7	40.82.159.29	20001:30150/TCP

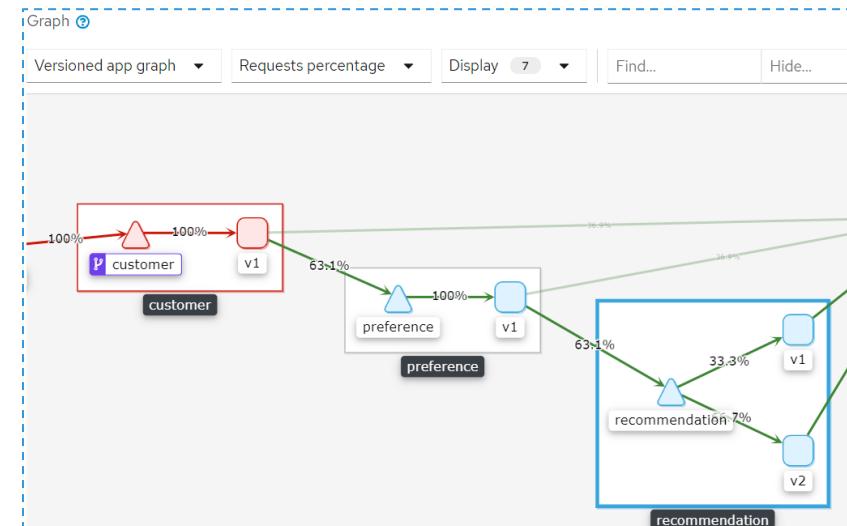
Lab. Istio – Simple Routing (1/2)

- Istio의 Simple Routing 확인
 - recommendation 서비스 추가 배포 (version. two)
 - kubectl apply -f <(istioctl kube-inject -f recommendation/kubernetes/Deployment-v2.yaml) -n tutorial
 - 서비스 호출
 - 브라우저에서 Customer 서비스(External-IP:8080 접속) 호출
 - F5(새로고침)를 10회 이상 클릭하여 다수의 요청 생성
- Routing 결과 확인
 - Kiali(External-IP:20001) 접속
 - Left 영역 : Graph 선택
 - Main 영역 : Versioned app > Requests percentage
 - Recommendation 서비스의 v1, v2에 균등한 라우팅 (Round Robin) 비율 확인 - 52.9% : 48.1%



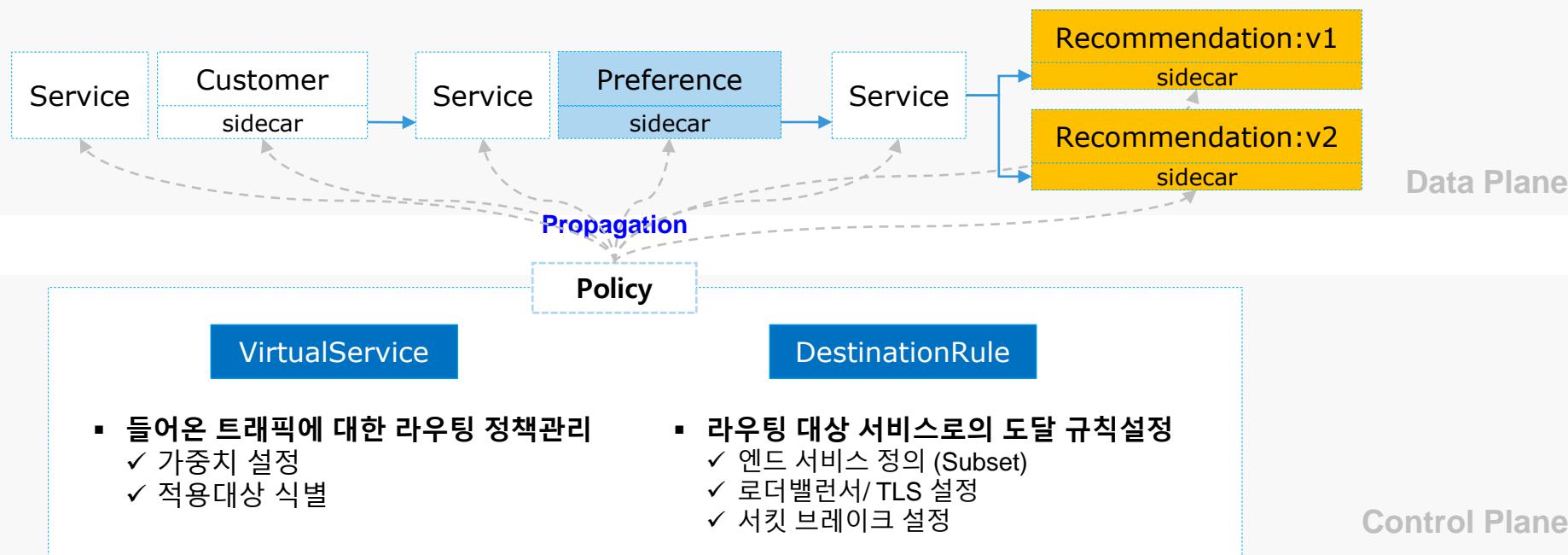
Lab. Istio – Simple Routing (2/2)

- recommendation 서비스 Scale Out
 - 서비스의 v2 의 replica 를 2로 설정
 - kubectl scale --replicas=2 deployment/recommendation-v2 -n tutorial
 - kubectl get po -n tutorial
 - 서비스 호출
 - 브라우저에서 Customer 서비스(External-IP:8080 접속) 호출
 - F5(새로고침)를 10회 이상 클릭하여 다수의 요청 생성
- Routing 결과 확인
 - Kiali(External-IP:20001) 접속
 - Left 영역 : Graph 선택
 - Main 영역 : Versioned app > Requests percentage
 - Recommendation 서비스의 v1, v2에 균등한 라우팅 (Round Robin) 비율 확인 - 33.3% : 66.6%



Istio Improved Routing & Rule

- 이스티오 정책 : VirtualService, DestinationRule



Lab. Istio – Improved Routing (1/3)

- 정책(VirtualService, DestinationRule) 설정 실습
 - 현재 정책 확인
 - kubectl get VirtualService -n tutorial -o yaml, kubectl get DestinationRule -n tutorial -o yaml

recommendation 서비스 라우팅 정책 설정

VirtualService/ DestinationRule 설정

- kubectl create -f istiofiles/destination-rule-recommendation-v1-v2.yaml -n tutorial
- kubectl create -f istiofiles/virtual-service-recommendation-v2.yaml -n tutorial

설정 확인

- kubectl get VirtualService -n tutorial -o yaml

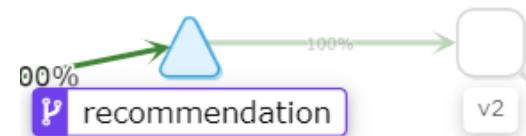
```
spec:  
  hosts:  
    - recommendation  
  http:  
    - route:  
      - destination:  
          host: recommendation  
          subset: version-v2  
          weight: 100
```

- kubectl get DestinationRule -n tutorial -o yaml

```
spec:  
  host: recommendation  
  subsets:  
    - labels:  
        version: v1  
        name: version-v1  
    - labels:  
        version: v2  
        name: version-v2
```

서비스 확인

- 브라우저에서 Customer 서비스(External-IP:8080 접속)호출
- Kiali(External-IP:20001)에서 모니터링

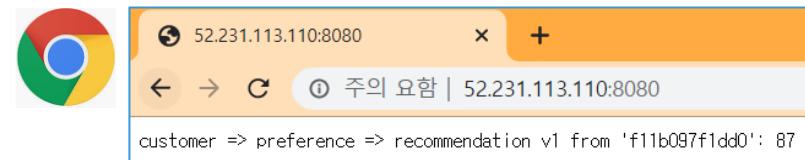
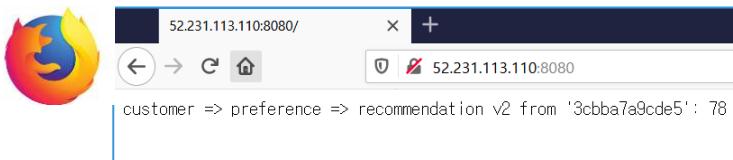


Lab. Istio – Improved Routing (2/3)

- 가중치 기반 Service Routing 실습
 - recommendation 서비스 v1의 가중치를 100으로 변경
 - kubectl replace -f istiofiles/virtual-service-recommendation-v1.yml -n tutorial
 - 서비스 호출 및 Kiali(Externl-IP:20001)에서 모니터링
- VirtualService 삭제 시, Round-Robin 방식으로 동작
 - kubectl delete -f istiofiles/virtual-service-recommendation-v1.yml -n tutorial
- 이와 같이 가중치를 사용해 카나리 배포(Canary Deploy)가 가능함
 - 90 : 10
 - kubectl create -f istiofiles/virtual-service-recommendation-v1_and_v2.yml -n tutorial
 - 75 : 25
 - kubectl replace -f istiofiles/virtual-service-recommendation-v1_and_v2_75_25.yml -n tutorial
- 삭제
 - kubectl delete dr recommendation -n tutorial
 - # kubectl delete vs recommendation -n tutorial
 - kubectl scale --replicas=1 deployment/recommendation-v2 -n tutorial

Lab. Istio – Improved Routing (3/3)

- Client 브라우저 유형별 Service Routing 실습
 - Firefox 브라우저로 접속 시, v2로 라우팅되도록 설정
 - kubectl create -f istiofiles/destination-rule-recommendation-v1-v2.yml -n tutorial
 - kubectl create -f istiofiles/virtual-service-firefox-recommendation-v2.yml -n tutorial
 - Firefox 브라우저와 다른 브라우저에서 접속 확인
 - Browser 환경이 지원되지 않을 경우,
 - curl -A Safari External-IP:8080
 - curl -A Firefox External-IP:8080

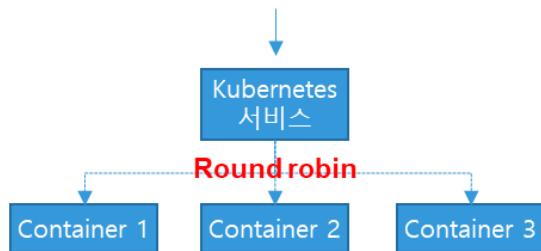


- 삭제
 - kubectl delete dr recommendation -n tutorial
 - kubectl delete vs recommendation -n tutorial

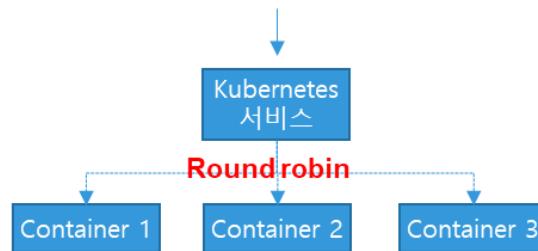
Lab. Istio – Circuit Breaker

- Pre-Requisite : Lab에 적용된 Customized Spring Actuator에 대한 이해

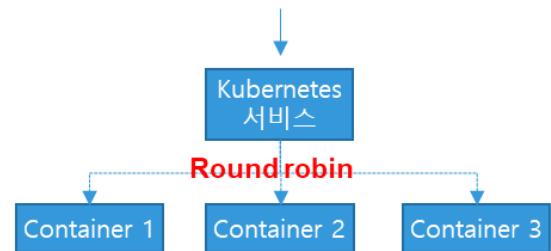
GET http://delivery:8080/actuator/echo



PUT http://delivery:8080/actuator/down



GET http://delivery:8080/actuator/health



- 어느 한 컨테이너의 ID와 IP반환

- 어느 한 컨테이너의 actuator 상태를
‘503(Down)’ 상태로 변경

- 어느 한 컨테이너의 actuator 상태를 리턴
- Istio의 Circuit Breaker가 적용되어 있는 경우, 5xx코드를 리턴하는 컨테이너를 자동으로 Pool Ejection 처리하고 Retry

Lab. Istio



Lab Time

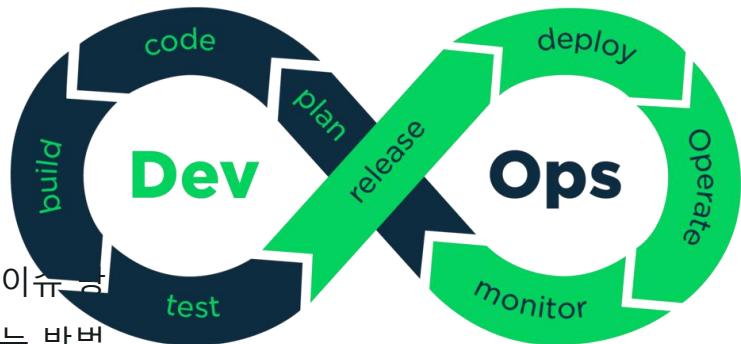
- Lab Script Location
 - Workflowy :

Table of content

1. Service Mesh: Istio
2. DevOps Process and Container-based Pipeline 
3. Version Control(SCM) & Build Automation Tools
4. AWS CodeStar
5. AWS CodeBuild

What's CI/CD? DevOps? Pipeline?

- Continuous Software Development Life Cycle in DevOps
- CI/CD 파이프라인은 새 버전의 소프트웨어를 제공하기 위해 수행해야 할 일련의 단계



- 개발자가 코드를 지속적으로 배포할 수 있어야 하는 필요성
- 초기 단계에서 버그를 감지하고 빈번한 코드 커밋으로 인한 이슈 등
- CI/CD는 어플리케이션을 보다 짧은 주기로 고객에게 제공하는 방법
- 어플리케이션의 통합 및 테스트 단계에서부터 제공 및 배포에 이르는 어플리케이션의 라이프사이클 전체에 걸쳐 자동화와 모니터링을 제공하는 프로세스들의 묶음을 의미

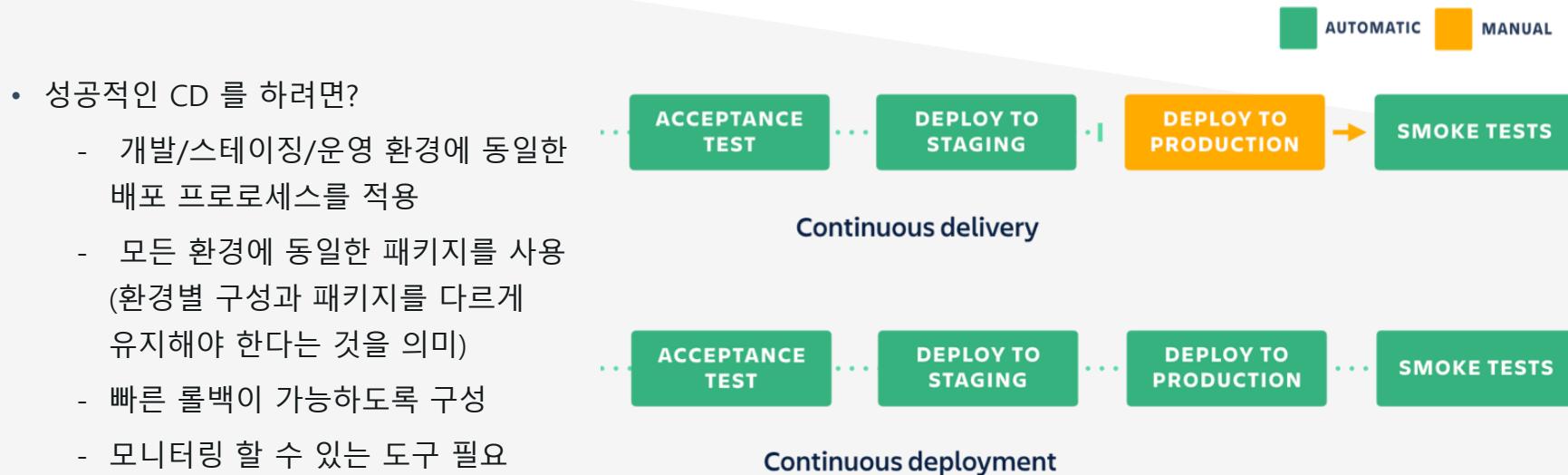
Continuous Integration

- 개발자를 위한 자동화 프로세스인 지속적인 통합(Continuous Integration)을 의미
- **개발코드를 통합할 때의 문제점을 해결하고,
자동화시켜 지속적으로 유지시키는 방법**
- 코드를 커밋에 따른 빌드, 통합, 테스트 과정의 자동화
- 성공적인 CI 를 하려면?
 - 코드 저장소에 모든것을 넣어야 합니다.
 - 코드는 자주 병합되어야 합니다.
 - 매일 빌드를 성공적으로 실행해야 합니다.
 - 빌드 프로세스는 완전 자동화 되어야 합니다.
 - 빌드 실패시 바로 수정이 되어야 합니다.
 - 빌드에 번호가 매겨지고, 반복 가능해야 합니다.
 - CI 결과물로 만들어진 패키지 혹은 컨테이너는 신뢰 할 수 있어야 합니다. (테스트 자동화 필수)



Continuous Delivery / Continuous Deployment

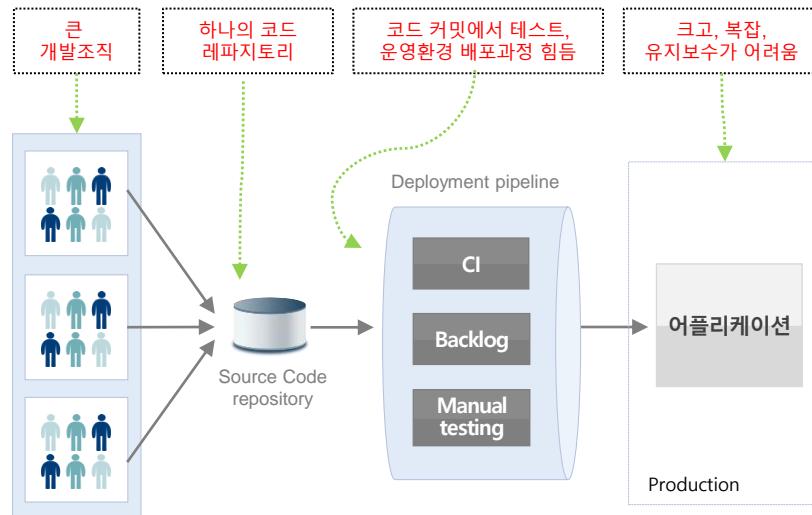
- 지속적인 서비스 제공(Continuous Delivery) / 지속적인 배포(Continuous Deployment)
- 어플리케이션을 항상 신뢰 가능한 수준으로 배포 될수 있도록 지속적으로 관리
- CI 가 이루어지고 난 후에 운영환경 까지 배포를 수행하여,
실제 사용자가 사용할 수 있도록 적용하는 단계**



Process Change : 열차말고 택시를 타라!

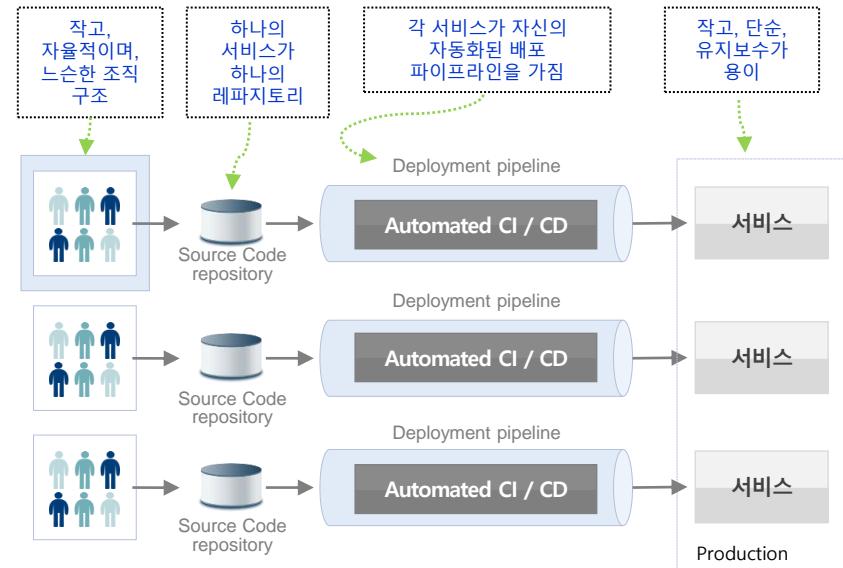
모노리식 개발 및 운영환경

- 모노리식 환경하에서는 큰 개발팀이 하나의 소스코드 레파지토리에 변경 사항을 커밋하므로 코드간 상호 의존도가 높아 신규 개발 및 변경이 어려움
- 작은 변화에도 전제를 다시 테스트/ 배포하는 구조이므로 통합 스케줄에 맞춘 파이프라인을 적용하기가 어렵고 Delivery 시간이 과다 소요



マイ크로서비스 개발 및 운영환경

- 작고 분화된 조직에서 서비스를 작은 크기로 나누어 개발하므로 해당 비즈니스 로직에만 집중하게 되어 개발 생산성 향상
- 연관된 마이크로서비스만 테스트를 수행하므로 개발/테스트/배포 일정이 대폭 축소



CI/CD Tools - Open Source



Jenkins

- ✓ 무료 및 오픈 소스로 연동 가능한 다양한 플러그인 (1,500개 이상) 제공
- ✓ AWS, Google Cloud, Azure, Digital Ocean 등 퍼블릭 클라우드 플랫폼과 통합
- ✓ 병렬 작업을 수행하고 복잡한 CD 요구 사항을 실현하는 데 활용
- ✓ 독립 실행형 Java 응용 프로그램이며 즉시 사용할 수 있는 .war 형식

적은 레퍼런스, 러닝 커브
빌드 구성은 100개까지
무료

- ✓ 확장성이 뛰어나며, 병렬 빌드가 가능하고 다른 빌드와 환경에서 동시 빌드 실행
- ✓ Kotlin 기반으로 Docker, Visual Studio Team Services, Maven, NuGet 등과 통합
- ✓ Google Cloud, AWS, VMWare vSphere 클라우드 플랫폼과 통합 가능

TeamCity

circleci

- ✓ CI/CD 파이프라인을 "워크플로"를 제공하며 CircleCI의 병렬 실행 지원
- ✓ CircleCI의 CircleCI Server는 GitHub Enterprise, LambdaTest, Coveralls 통합 지원
- ✓ AWS, Google Cloud, Azure 등 퍼블릭 클라우드 플랫폼 연동 지원

Open Source 프로젝트에
대해서만 무료, Hosted 방식

- ✓ Travis CI는 Java, C#, Julia, Python 등 프로그래밍 언어(즉, 총 30개) 지원
- ✓ CI/CD는 GitHub Enterprise 도구와 원활하게 통합되는 독점 YAML 구문을 사용
- ✓ AWS, Google Cloud, Azure 등 퍼블릭 클라우드 플랫폼 연동 지원



Travis CI

Bamboo

- ✓ Jira 소프트웨어 및 Bitbucket 서버와의 통합 기능 내장
- ✓ Bamboo는 Docker, AWS 플랫폼 통합
- ✓ 최대 100개의 원격 빌드 에이전트와 에이전트에 대한 병렬 테스트 배치 지원

상용 SW

CI/CD Tools - Cloud Vendors'



AWS CodeBuild

- ✓ CI/CD(지속적 통합 및 전달)를 위한 완전한 자동 소프트웨어 릴리스 워크플로를 생성 가능
- ✓ Jenkins 서버 설정에서 CodeBuild를 작업자 노드로 사용하여 빌드를 분산
- ✓ 빌드 볼륨에 따라 자동으로 확장 및 축소
- ✓ 빌드를 완료할 때까지 걸리는 시간(분)을 기준으로 비용이 청구

- ✓ Azure Pipeline에 있는 DevOps는 프로젝트 시작에서 기획, 작업관리, 모니터링 도구포
- ✓ 부하 테스트를 포함하는 통합 테스트 및 모니터링 환경 제공
- ✓ DevOps를 위해 CI/ CD(Continuous Deployment(Delivery))를 분리하여 구축
- ✓ CI와 CD trigger 설정 및 파이프라인 자동화



Azure Pipelines

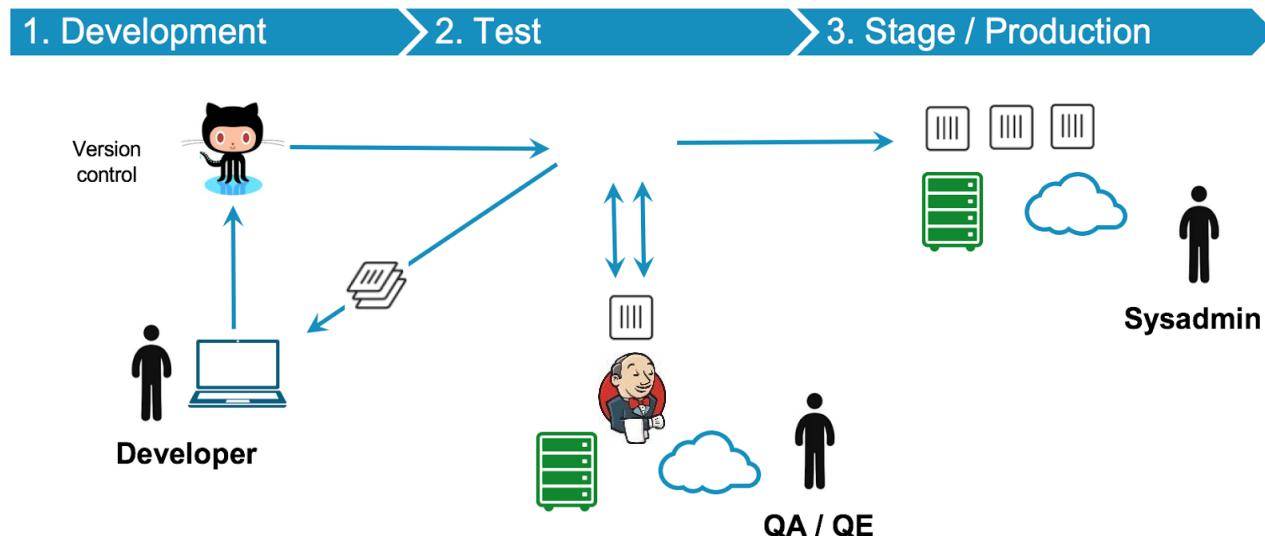


Google Cloud Build

- ✓ Cloud Build는 Google Cloud Platform의 인프라에서 빌드를 실행하는 서비스
- ✓ 매일 120분의 무료 빌드와 최대 10회 동시 빌드 지원
- ✓ Docker, gradle, maven, bazel 같은 빌드 도구로 artifact 생성 가능
- ✓ 컨테이너 이미지 패키지 취약점 자동분석

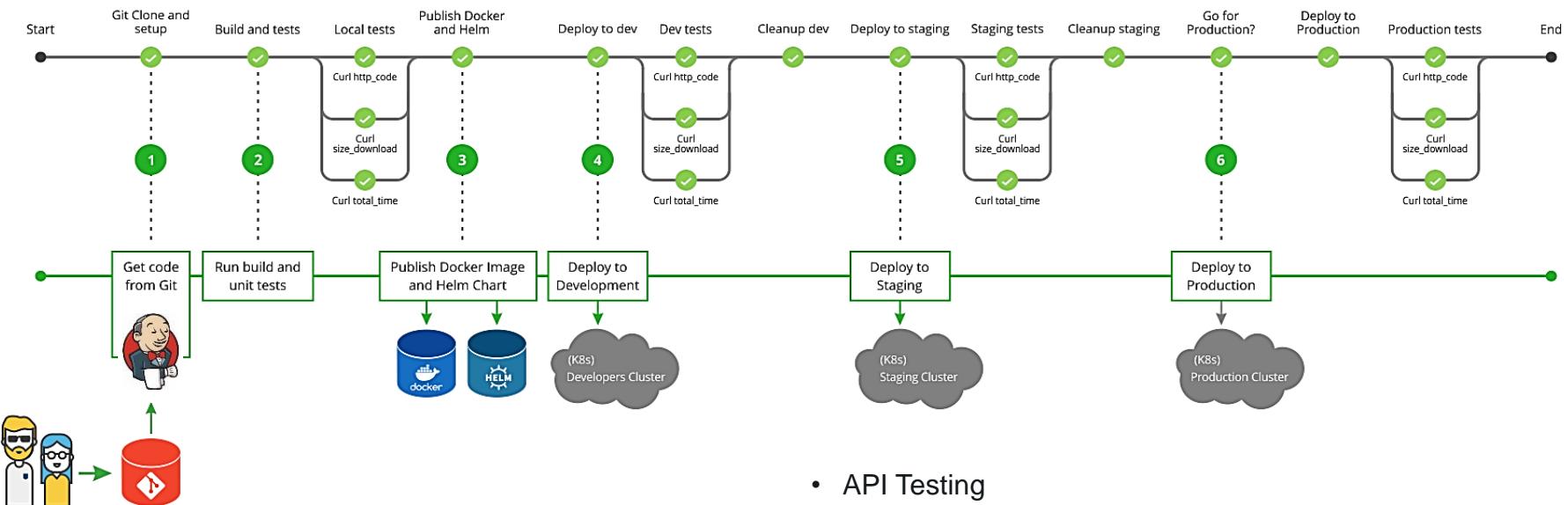
CI/CD Pipeline – conventional CI/CD

CI /CD Workflow



- 잦은 배포
- 일반 서버에 배포
- 테스트 자동화/커버리지
- 도구: Jenkins, Travis, Gitlab 등

CI/CD Pipeline – Container-based



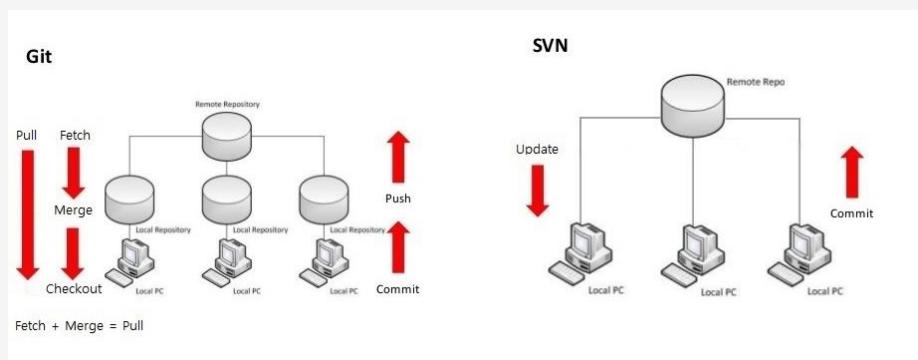
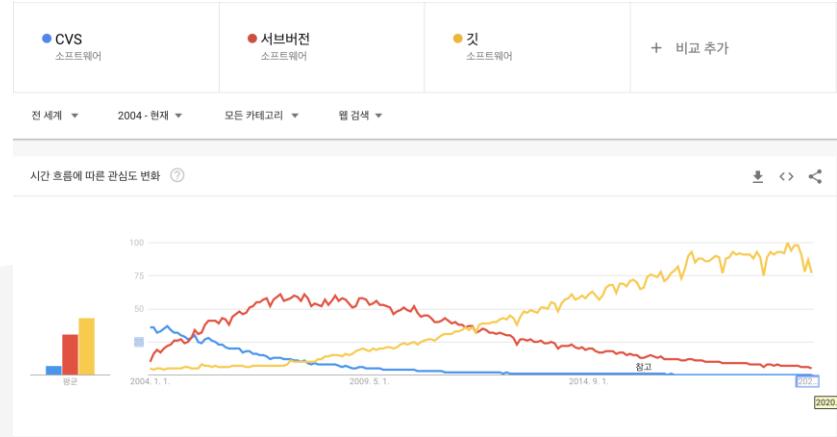
- API Testing
- Blue/Green, Canary
- 도구 : Jenkins + Docker + Spinnaker + Helm + Kubernetes
→ 매우복잡

Table of content

1. Service Mesh: Istio
2. DevOps Process and Container-based Pipeline
3. Version Control(SCM) & Build Automation Tools 
4. AWS CodeStar
5. AWS CodeBuild

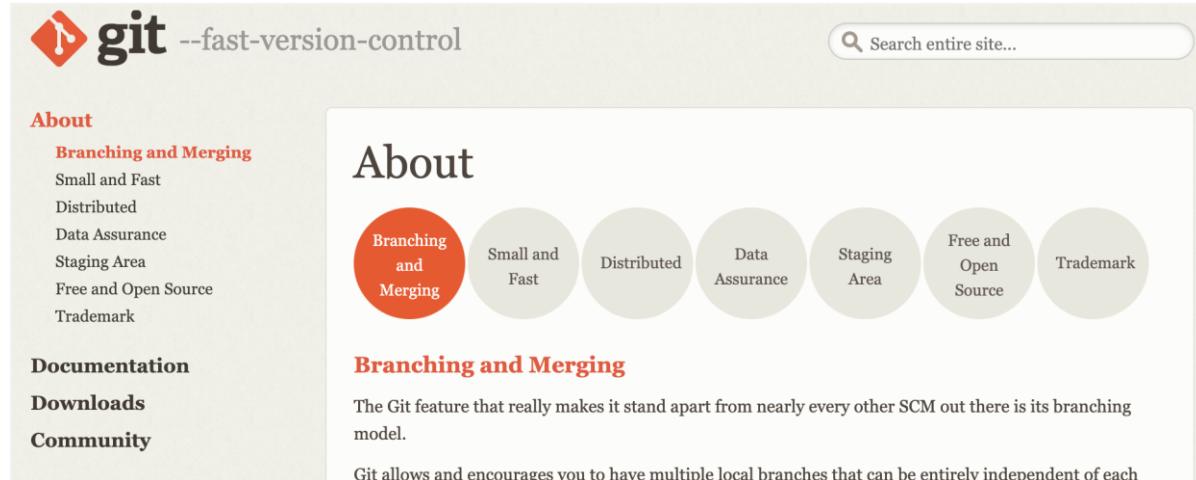
형상 관리도구 - History

- 형상관리란? : 소스의 변화를 끊임없이 추적하고, 버전별로 관리
- CVS(Concurrent version system)
 - 1980년대에 만들어진 형상관리 툴이지만 파일 관리나 커밋 중 오류 시 롤백이 되지 않는 등 불편한 문제점이 있어 이후 SVN으로 대체됨
- SVN (subversion)
 - 2000년에 CVS를 대체하기 위해 만들어졌음
 - Trunk, Tag, Branch 구조를 사용
 - 중앙 레파지토리 방식
 - 개발자가 자신만의 version history를 가질 수 없음
- GIT
 - 2005년 리누스 토팔즈에 의해 시작
 - 매우 빠른 속도와 분산형 저장소. SVN보다 많은 기능을 지원
 - 개발자가 자신만의 commit history를 가질 수 있음
 - 저장소 분리로, 복원이 용이



형상 관리도구 - GIT

1. Branch and Merge : 새로운 기능 패치시 브랜치를 생성하고 다시 메인코드로 병합 가능
2. Small and Fast : 로컬에서 우선작업하여 빠름
3. 분산형 데이터 모델
4. 데이터 안전 : 모든 파일 체크섬 검사
5. 스테이징 모드 : local repo 와 remote repo 로 2단계 저장소를 가짐



The screenshot shows the official Git website at <https://git-scm.com/>. The top navigation bar includes links for Home, Documentation, Downloads, and Community. The main content area features the Git logo and the tagline "fast-version-control". A search bar is located in the top right corner. The "About" section is highlighted in red. Below it, there's a list of features: Branching and Merging, Small and Fast, Distributed, Data Assurance, Staging Area, Free and Open Source, and Trademark. To the right of the "About" section, there's a grid of circular icons representing these features. The "Branching and Merging" icon is highlighted in red, while the others are grey.

About

- Branching and Merging
- Small and Fast
- Distributed
- Data Assurance
- Staging Area
- Free and Open Source
- Trademark

Documentation

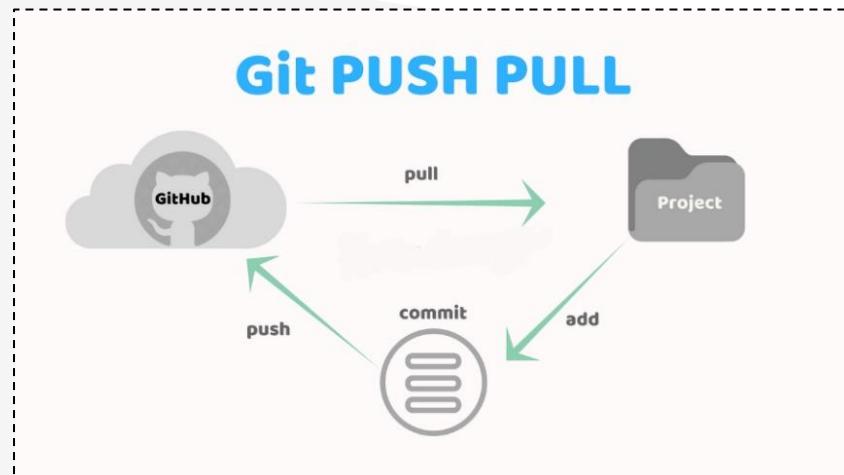
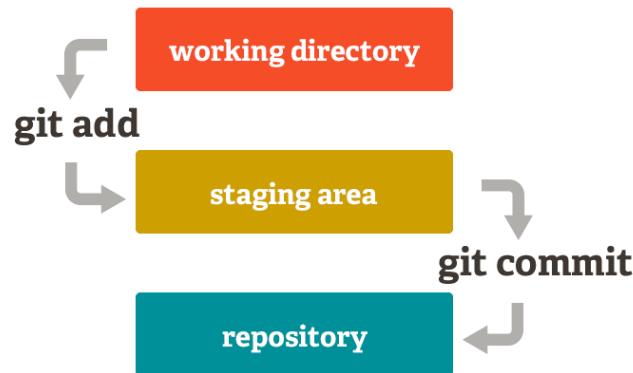
Downloads

Community

<https://git-scm.com/about/>

Git 사용해 보기

- github.com에서 repository 생성
- git init
- git config
 - git config --global user.name
 - git config --global user.email
- git status
- git add .
- git commit -m "1st commit"
- git remote
- git clone
- git pull
- git push



빌드 자동화 도구

- 빌드 자동화란? : 자바 소스를 compile하고 package해서 deploy하는 일을 자동화 해주는 것
- Apache Ant
 - Another Neat Tool
 - 2000년 출시
 - Base build file : build.xml
 - 유연함이 장점이지만 (모든 명령을 직접작성), 규칙이 없기 때문에 유지보수가 어려움
- Apache Maven
 - Ant의 불편함을 해소하고자 2004년 출시
 - 규칙을 정하고 Goals라는 사전 정의된 command를 제공
 - Base build file : pom.xml
- Gradle
 - Ant와 Maven의 장점을 모아 2012년 출시
 - Android OS의 빌드 도구로 채택
 - 프로그래밍 언어 형식으로 유연함이 장점 (groovy 파일로 작성)
 - Base build file : build.gradle

“Build Automation”

Maven Core

- Plugin
- Lifecycle
- Dependency
- Profile
- POM

Plugin

- 메이븐은 플러그인을 구동해주는 프레임워크(plugin execution framework)이다. 모든 작업은 플러그인에서 수행한다.
- 플러그인은 다른 산출물(artifacts)와 같이 저장소에서 관리된다.
- 메이븐은 여러 플러그인으로 구성되어 있으며, 각각의 플러그인은 하나 이상의 goal(명령, 작업)을 포함하고 있다. Goal은 Maven의 실행단위
- 플러그인과 골의 조합으로 실행한다. ex. mvn <plugin>:<goal> = mvn spring-boot:run
- 메이븐은 여러 Goal을 묶어서 Lifecycle phases 로 만들고 실행한다. ex. mvn <phase> = mvn install

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

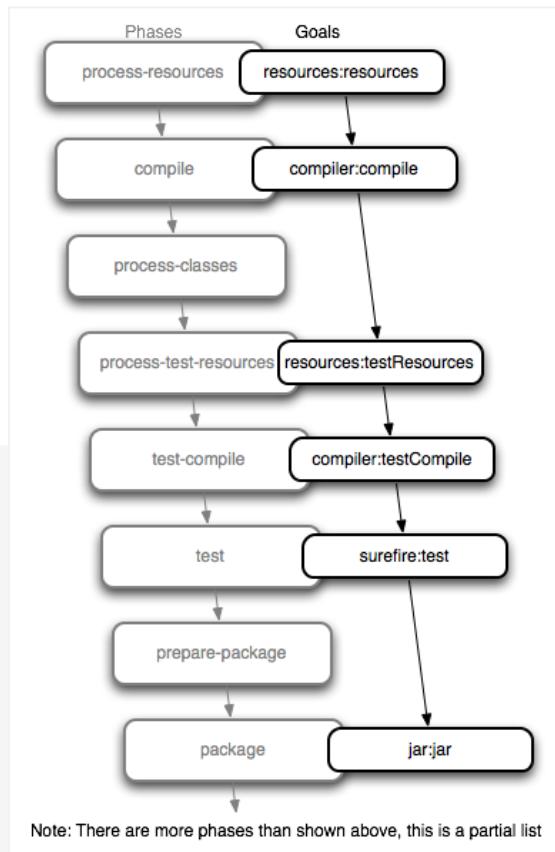
Lifecycle

Phase

Plug-in

Goal

- 메이븐 프로젝트 생성에 필요한 단계들 (phases) 의 묶음
- clean, default, site 세 가지로 표준 정의
- clean : 빌드 시 생성되었던 산출물을 삭제
- default : 프로젝트 배포절차, 패키지 타입별로 다르게 정의
- site : 프로젝트 문서화 절차
- mvn compile : Java 소스를 컴파일하여 target 디렉토리에 생성
- mvn test : 생성된 test 코드를 실행하여 target 디렉토리에 생성
- **mvn package** : target 디렉토리 하위에 jar, war, ear 등 패키지 파일을 생성
- mvn install : 로컬 저장소로 배포
- mvn deploy : 원격 저장소로 배포



Dependency

- 라이브러리 다운로드 자동화
- 참조하고 있는 library 까지 모두 찾아서 추가 (의존성 전이)
- USER_HOME/.m2/repository 에 저장
- 의존관계 제한 기능
 - Scope
 - compile : 기본값, 모든 classpath 에 추가
 - provided : 컴파일시 필요하나 실행때는 필요없음
 - runtime : 실행때는 필요하나 컴파일때는 필요없음
 - test : 테스트때만 사용
 - system : jar 파일을 직접 지정
- Dependency management : 직접 참조하지는 않으면서 하위 모듈이 특정 모듈을 참조할 경우, 특정 모듈의 버전을 지정 (예 : spring-cloud)
- Excluded dependencies : 임의의 모듈에서 참조하는 특정 하위 모듈을 명시적으로 제외처리 (예 : log)

```

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <exclusions>
    <exclusion>

      <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
logging</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-log4j2</artifactId>
  </dependency>

```

Profile

- Local , develop, test, production 등 환경에 따라 달라져야할 정보들을 Build 타임에 구성 한다.
- P 옵션으로 프로파일을 선택하여 build
(mvn package -P dev)
- 환경마다 build 구성이 필요하므로 **Spring Profile** 을 권장

```
<profiles>
  <profile>
    <id>dev</id>
    <properties>
      <env>dev</env>
    </properties>
  </profile>
  <profile>
    <id>test</id>
    <properties>
      <env>test</env>
    </properties>
  </profile>
</profiles>
```

POM.xml

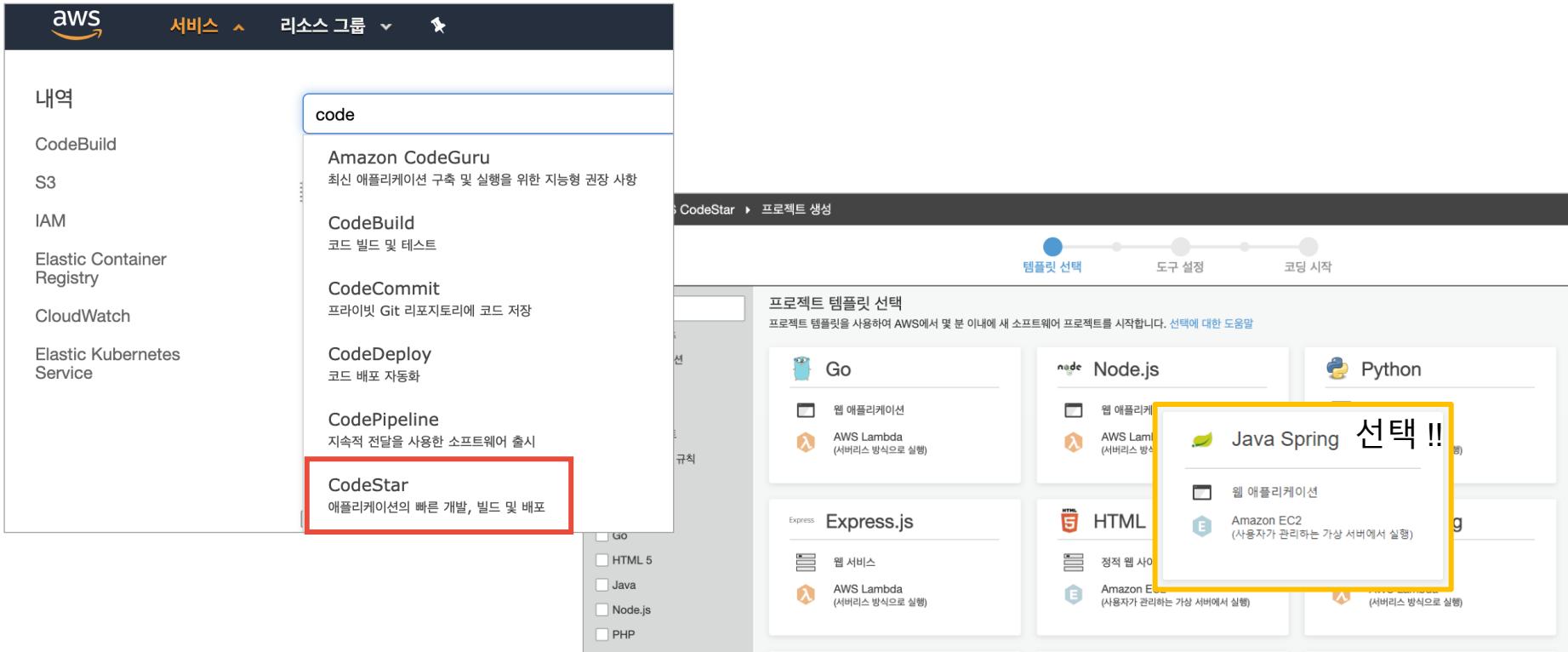
- POM은 프로젝트 객체 모델(Project Object Model)
 - 프로젝트 당 1개
 - 프로젝트의 root에 존재
 - <groupId>, <artifactId>, <version>으로 자원을 식별
-
- <groupId> : 프로젝트의 패키지 명칭
 - <artifactId> : artifact 이름, groupId 내에서 유일해야 한다.
 - <packaging> : 패키징 유형(jar, war 등)
 - <distributionManagement> : artifact가 배포될 저장소 정보와 설정
 - <dependencyManagement> : 의존성 처리에 대한 기본 설정 영역
 - <dependencies> : 의존성 정의 영역
 - <repositories> : default 는 공식 maven repo
 - <build> : 빌드에 사용할 플러그인 목록을 나열

Table of content

1. Service Mesh: Istio
2. DevOps Process and Container-based Pipeline
3. Version Control(SCM) & Build Automation Tools
4. AWS CodeStar ✓
5. AWS CodeBuild

AWS DevOps – CodeStar

- 웹 어플리케이션 템플릿을 사용, AWS에서 애플리케이션을 빠르게 개발, 빌드, 배포할 수 있는 통합 도구



AWS DevOps – CodeStar

- 웹 어플리케이션 템플릿을 사용, AWS에서 애플리케이션을 빠르게 개발, 빌드, 배포할 수 있는 통합 도구

The image shows the AWS CodeStar project creation interface and its resulting dashboard.

Project Creation Interface:

- Step Progress:** 템플릿 선택 (Template Selection) is completed, while 도구 설정 (Tool Configuration) and 코딩 시작 (Coding Start) are in progress.
- Project Information:** The project name is "sample-codestar". A yellow box highlights the input field for the Project ID, with the placeholder "sample-codestar" and the instruction "user00-codestar 입력 !!".
- Source Options:** AWS CodeCommit and GitHub are listed as options for source code storage.
- Repository Name:** The repository name is "sample-codestar".
- Next Step Buttons:** "이전" (Previous) and "다음" (Next).

Resulting Dashboard:

- Dashboard Header:** AWS CodeStar > sample-codestar
- Left Sidebar:** 대시보드, 코드, 빌드, 배포, 파이프라인, 팀, 확장.
- Project Status:** AWS CodeStar 프로젝트가 성공적으로 생성되었습니다.
- Callout Message:** sample-codestar에 오신 것을 환영합니다! 시작하기를 도와드립니다.
- Thumbnail Preview:** A screenshot of the AWS CodeStar dashboard showing monitoring and build logs.
- Bottom Callout:** 개발툴 건너뛰기 및 설정완료 (Skip Development Tools and Set Up Configuration).

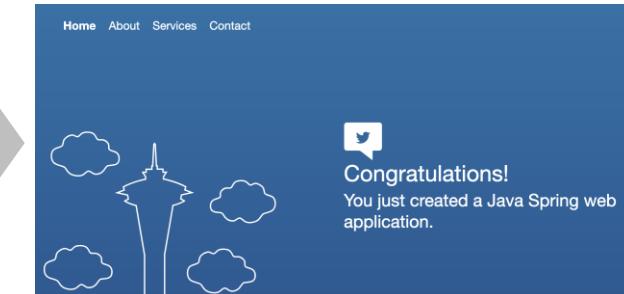
AWS DevOps – CodeStar

- 웹 어플리케이션 템플릿을 사용, AWS에서 애플리케이션을 빠르게 개발, 빌드, 배포할 수 있는 통합 도구

The screenshot shows the AWS CodeStar console. On the left, a sidebar navigation includes: 대시보드, 코드, 빌드, 배포, 파일프赖인, 팀, 확장, and 프로젝트. The main area displays:

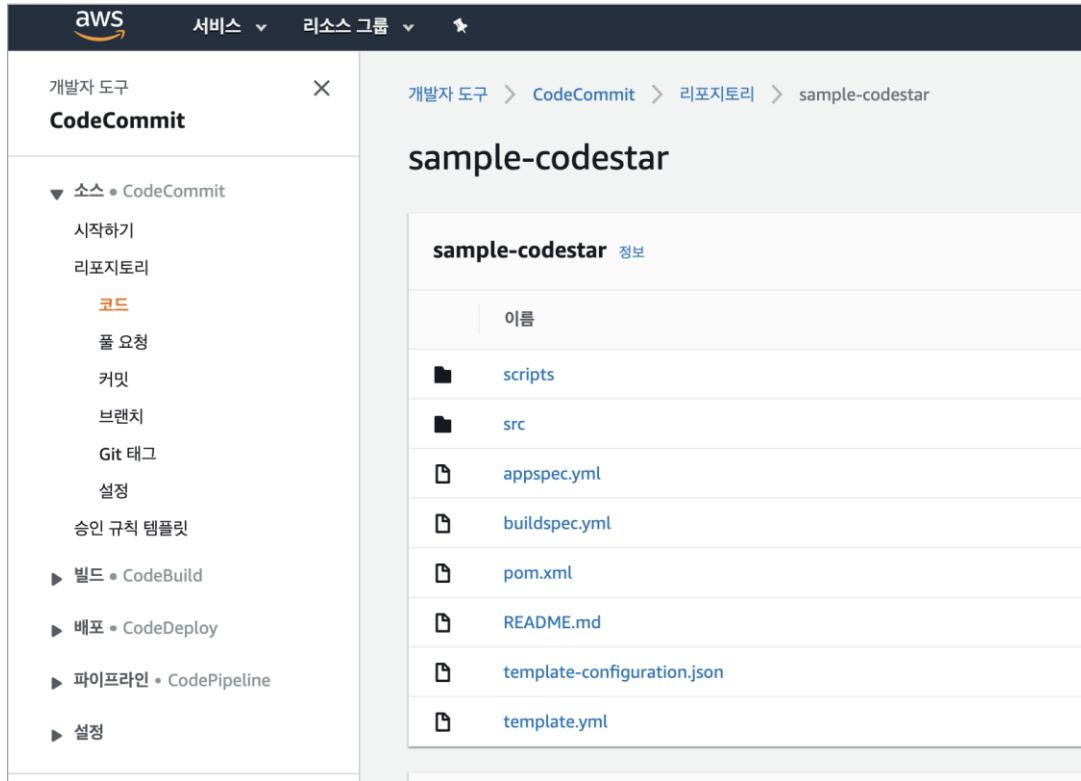
- 5. 프로젝트 대시보드 사용자 정의**: Includes links to the AWS CodeStar 설명서 보기 and AWS CodeStar 포럼 방문.
- 커밋 이력: sample-codestar**: Shows two commits: "readme" by kimscott at 1 minute ago and "Initial commit by AWS CodeCommit" by AWS CodeCommit at 1 hour ago.
- 연결**: AWS CodeCommit 세부 정보.
- 애플리케이션 활동**: A chart showing application activity over time, with a sharp spike at 12:00 on March 19, 2020, labeled "Amazon CloudWatch".
- 파이프라인**: A detailed view of the pipeline stages:
 - Source**: 2020. 3. 19. 오후 12:52:27 ApplicationSource CodeCommit 성공. Includes a "커밋 이력" section.
 - Build**: 2020. 3. 19. 오후 12:52:27 PackageExport CodeBuild 진행 중. Includes a "Build Log" section.
 - Deploy**: 2020. 3. 19. 오전 11:57:43 GenerateChangeSet CloudFormation 성공. Includes a "배포 이력" section.AWS CodePipeline 세부 정보 is also shown at the bottom.

1. 자동으로 커밋-빌드-배포로 이루어진 파이프라인이 생성됨
2. EC2 인스턴스를 생성하여 어플리케이션을 자동 배포함



AWS DevOps – CodeCommit

- AWS가 Managed Service로 제공하는 프라이빗 Git 리파지토리



1. Private Repository
2. Git 사용
3. 사용하기 위해서는 User에 권한을 부여해 주어야 함
4. 권한부여 iam 주소
<https://console.aws.amazon.com/iam/>
5. buildspec.yml – 코드 빌드에 사용되는 파일
6. appspec.yml – 코드 디플로이에 사용되는 파일

AWS DevOps – CodeCommit

- AWS가 Managed Service로 제공하는 프라이빗 Git 리파지토리

Identity and Access Management(IAM)

- 대시보드
- 액세스 관리
- 그룹
- 사용자
- 역할
- 정책
- 자격 증명 공급자
- 계정 설정
- 보고서 액세스
- 액세스 분석기

권한 그룹 (1) 태그 (1) 보안 자격 증명 액세스 관리자

AWS CodeCommit에 대한 HTTPS Git 자격 증명

AWS CodeCommit 리포지토리에 대한 HTTPS 연결을 인증하는데 사용할 수 있는 사용자 이름과 암호를 생성합니다. 자격 증명 세트를 최대 100개까지 만들 수 있습니다.

작업 ▾

사용자 이름	상태	생성 완료
uengineDev-at-979050235289	(활성)	2020-03-19 12:36 UTC+0900

```
→ MSA project git clone https://git-codecommit.ap-northeast-1.amazonaws.com/  
  
Cloning into 'sample-codestar'...  
Username for 'https://git-codecommit.ap-northeast-1.amazonaws.com': uengineDev  
Password for 'https://uengineDev-at-979050235289@git-codecommit.ap-northeast-1.amazonaws.com':  
remote: Counting objects: 44, done.  
Unpacking objects: 100% (44/44), done.  
→ MSA project cd sample-codestar  
→ sample-codestar git:(master) █
```

1. IAM 의 사용자 – 보안 자격 증명에서 CodeCommit 자격 생성
2. ID / PW 가 발급되면 해당 계정으로 CodeCommit의 git 접속 가능

AWS DevOps – CodeDeploy

- AWS가 Managed Service로 제공하는 배포 자동화 도구 (**‘20. 12 현재, EKS 배포 미지원**)

The screenshot shows the 'Application Creation' step of the AWS CodeDeploy wizard. It includes fields for the application name ('sample-deploy'), deployment group ('EC2/OnPremises'), and a note about the deployment group being required. At the bottom, there are 'Cancel' and 'Create Application' buttons.

개발자 도구 > CodeDeploy > 애플리케이션 > 애플리케이션 생성

애플리케이션 생성

애플리케이션 구성

애플리케이션 이름
애플리케이션 이름을 입력합니다.
sample-deploy
100자 이내

컴퓨팅 플랫폼
컴퓨팅 플랫폼 선택
EC2/온프레미스

취소 **애플리케이션 생성**

1. 배포 어플리케이션 – 배포 그룹 생성 순으로 진행
2. 배포 그룹에서 블루/그린 디플로이가 가능함
3. 아직 EKS를 지원 안함

The screenshot shows the 'Application details' page for 'sample-deploy'. It lists the application name ('sample-deploy') and computing platform ('EC2/OnPremises'). Below this, the 'Deployment Groups' tab is selected, showing a table with one entry: 'Deployment Group' (empty). A note at the bottom states that CodeDeploy requires a deployment group to be created before proceeding.

sample-deploy

Application details

이름	컴퓨팅 플랫폼
sample-deploy	EC2/온프레미스

배포 | **배포 그룹** | 개정

배포 그룹

이름	상태	최근 시도한 배포	최근 성공한 배포	트리거 개수

배포 그룹 없음
CodeDeploy를 사용하여 애플리케이션을 배포하기 전에 배포 그룹을 생성해야 합니다.

배포 그룹 생성

AWS DevOps – CodePipeline

- CI/CD의 풀 라이프사이클(코드, 빌드, 배포)을 관리하는 AWS 관리형 서비스

Step 1
파이프라인 설정 선택

Step 2
소스 스테이지 추가

Step 3
빌드 스테이지 추가

Step 4
배포 스테이지 추가

Step 5
검토

파이프라인 설정 선택

파이프라인 설정

파이프라인 이름
파이프라인 이름을 입력합니다. 생성된 후에는 파이프라인 이름을 편집할 수 없습니다.
 100자를 초과할 수 없습니다.

서비스 역할
 새 서비스 역할
계정에 서비스 역할 생성 기존 서비스 역할
계정에서 기존 서비스 역할 선택

역할 이름

서비스 역할 이름 입력
 AWS CodePipeline이 이 새 파이프라인에 사용할 서비스 역할을 생성하도록 허용

- CodeStar를 사용하면 자동으로 파이프라인 생성
- 직접 만들기 위해서는 빌드/배포 단계를 생성해야 함

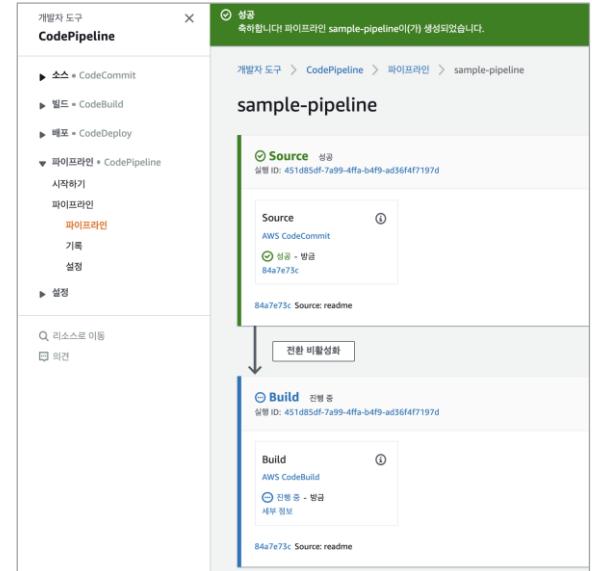


Table of content

1. Service Mesh: Istio
2. DevOps Process and Container-based Pipeline
3. Version Control(SCM) & Build Automation Tools
4. AWS CodeStar
5. AWS CodeBuild ✓

CodeBuild

이점

완전관리형 빌드 서비스

AWS CodeBuild는 기업이 자체적으로 서버 및 소프트웨어를 설정, 패치, 업데이트 및 관리할 필요성을 제거합니다. 설치 또는 관리가 필요한 소프트웨어가 없습니다.

확장 가능

사용자 지정 빌드 환경을 생성하면 CodeBuild에서 지원하는 사전 패키지 형태의 빌드 도구 및 런타임에 더해 자체 빌드 도구와 프로그래밍 런타임을 AWS CodeBuild에서 사용할 수 있습니다.

지속적 크기 조정

AWS CodeBuild는 빌드 볼륨에 따라 자동으로 확장 및 축소됩니다. 제출되는 빌드는 그때마다 즉각적으로 처리되며, 각 빌드를 동시에 실행할 수 있기 때문에 빌드가 대기열에 남겨지는 일이 없습니다.

지속적 통합 및 전달 지원

AWS CodeBuild는 AWS 코드 서비스 제품군에 속합니다. 즉, 이 서비스를 사용하여 CI/CD(지속적 통합 및 전달)를 위한 완전한 자동 소프트웨어 릴리스 워크플로를 생성할 수 있습니다. 또한 CodeBuild를 기준 CI/CD 워크플로에 통합하는 것도 가능합니다. 예를 들어 기존 Jenkins 서버 설정에서 CodeBuild를 작업자 노드로 사용하여 빌드를 분산할 수 있습니다.

사용량에 따라 지불

AWS CodeBuild에서는 빌드를 완료할 때까지 걸리는 시간(분)을 기준으로 비용이 청구됩니다. 이 말은 사용하지 않는 빌드 서버 용량에 대해서는 비용을 지불할 필요가 없다는 것을 의미합니다.

보안

AWS CodeBuild에서는 고객이 지정하고 AWS Key Management Service(KMS)로 관리되는 키를 사용해 빌드 애플리케이션이 암호화됩니다. CodeBuild는 AWS Identity and Access Management(IAM)에 통합되므로 사용자 지정 권한을 빌드 프로젝트에 할당할 수도 있습니다.

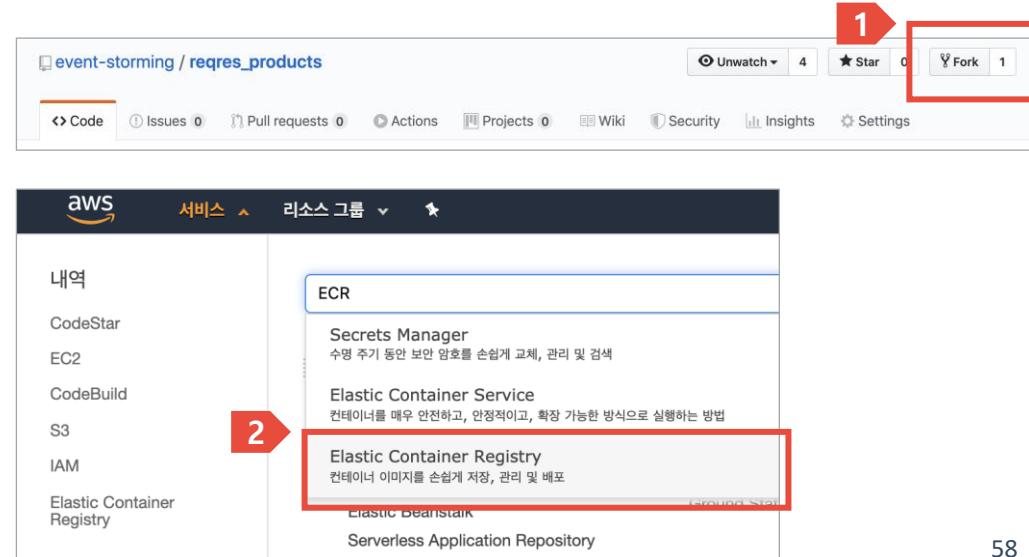
“

Let's Build a Microservice Pipeline

CodeBuild

- Github 의 소스를 빌드 / 패키징 하여 Docker Image 로 변경
- Docker Image 를 ECR 로 업로드
- 업로드된 Image 를 EKS(Amazon Elastic Kubernetes Service) 에 배포

1. Github 리파지토리 Fork
https://github.com/event-storming/reqres_products
2. Docker image 를 저장하기 위한 공간인 ECR 에 “user00-products” Repository 생성
3. Fork 받은 리파지토리의 Root에서 buildspec.yml을 편집하여 5행의 이미지명 수정



CodeBuild : 생성 설정 (1/3)

1. CodeBuild 프로젝트 생성

2. Github 계정 연결 후 Fork 한 리포지토리 연결

1

개발자 도구 > CodeBuild > 빌드 프로젝트 > 빌드 프로젝트 생성

빌드 프로젝트 생성

프로젝트 구성

프로젝트 이름

sample-products

프로젝트 이름은 2~255자여야 합니다. 글자(A-Z 및 a-z), 숫자(0~9) 및 특수 문자(- 및 _)를 포함할 수 있습니다.

설명 - 선택 사항

빌드 배치 - 선택 사항

빌드 배치 활성화

▶ 추가 구성

태그

2

소스

소스 1 - 기본

소스 공급자

GitHub

리포지토리

퍼블릭 리포지토리

내 GitHub 계정의 리포지토리

GitHub 리포지토리

https://github.com/kimscott/reqres_products.git

https://github.com/<user-name>/<repository-name>

연결 상태

OAuth를 사용하여 GitHub에 연결되었습니다.

GitHub에서 연결 해제

소스 버전 - 선택 사항 정보

풀 요청, 브랜치, 커밋 ID, 태그 또는 참조와 커밋 ID를 입력합니다.

▶ 추가 구성

Git clone 깊이, Git 하위 모듈

CodeBuild : 생성 설정 (2/3)

1. Webhook 을 설정하여 Github에 코드가 푸쉬될 때마다 트리거 동작
2. 빌드가 돌아갈 환경 설정
 - 운영체제 : Ubuntu
 - 런타임 : Standard
 - 이미지 : Standard4.0 (이미지별로 환경이 다르니 주의)
 - 환경유형 : Linux
 - 도커 권한 체크 필수

2

1

기본 소스 Webhook 이벤트 정보

필터 그룹 추가

Webhook - 선택 사항

코드 변경이 이 리포지토리에 푸시될 때마다 다시 빌드

한 개 이상의 Webhook 이벤트 필터 그룹을 추가하여 새 빌드를 트리거하는 이벤트를 지정합니다. Webhook 이벤트 필터 그룹을 추가하지 않은 경우에는 코드 변경이 리포지토리에 푸시될 때마다 새 빌드가 트리거됩니다.

Webhook 이벤트 필터 그룹 1

이벤트 유형

▶ 이러한 조건에서 빌드 시작

▶ 이러한 조건에서 빌드 시작 안 함

환경

환경 이미지

관리형 이미지
AWS CodeBuild에서 관리하는 이미지 사용

사용자 지정 이미지
도커 이미지 지정

운영체제

Ubuntu

① 이제 프로그래밍 언어 런타임은 Ubuntu 18.04의 표준 이미지에 포함됩니다. 이 이미지는 콘솔에서 생성된 새 CodeBuild 프로젝트에 대해 권장됩니다. 자세한 내용은 CodeBuild에서 제공하는 Docker 이미지 [문서](#)를 참조하십시오.

런타임

Standard

이미지

aws/codebuild/standard: 4.0

이미지 버전

이 런타임 버전에 항상 최신 이미지 사용

환경 유형

Linux

권한이 있음

도커 이미지를 빌드하거나 빌드의 권한을 승격하려면 이 플래그를 활성화합니다.

서비스 역할

새 서비스 역할
계정에 서비스 역할 생성

기존 서비스 역할
계정에서 기존 서비스 역할 선택

역할 이름

codebuild-sample-products-service-role

서비스 역할 이름 입력

▶ 추가 구성

제한 시간, 인증서, VPC, 컴퓨팅 유형, 환경 변수, 파일 시스템

CodeBuild : 생성 설정 (3/3)

1. 추가 구성에 환경 변수값을 입력 : AWS 계정 ID
 - AWS_ACCOUNT_ID
 - ECR에서 이미지 주소의 가장 앞에 값임
 - echo \$(aws sts get-caller-identity --query Account --output text) 명령으로 조회 가능
2. buildspec.yml 파일을 사용하겠다고 체크 후, 저장

2

Buildspec

빌드 사양

buildspec 파일 사용
YAML 형식의 buildspec 파일에 빌드 명령 저장

빌드 명령 삽입
빌드 명령을 빌드 프로젝트 구성으로 저장

Buildspec 이름 - 선택 사항
기본적으로 CodeBuild는 소스 코드 루트 디렉토리에서 buildspec.yml 파일을 찾습니다. buildspec 파일이 다른 이름 또는 위치를 사용하는 경우 여기에 소스 루트의 경로를 입력하십시오(예: buildspec-two.yml 또는 configuration/buildspec.yml).

아티팩트

아티팩트 1 – 기본

유형

아티팩트 없음

테스트를 실행하거나 도커 이미지를 Amazon ECR에 넣는 경우 [아티팩트 없음]을 선택할 수 있습니다.

▶ 추가 구성
캐시, 암호화 키

1

▼ 추가 구성
제한 시간, 인증서, VPC, 컴퓨팅 유형, 환경 변수, 파일 시스템

환경 변수

이름	값	유형
AWS_ACCOUNT_ID	97905023****	일반 텍스트 ▾
제거		

환경 변수 추가

CodeBuild : buildspec.yml

```
1 version: 0.2
2
3 env:
4   variables:
5     IMAGE_REPO_NAME: "products"
6
7 phases:
8   install:
9     runtime-versions:
10    java: corretto8
11    docker: 18
12   pre_build:
13     commands:
14       - echo Logging in to Amazon ECR...
15       - echo $IMAGE_REPO_NAME
16       - echo $AWS_ACCOUNT_ID
17       - echo $AWS_DEFAULT_REGION
18       - echo $CODEBUILD_RESOLVED_SOURCE_VERSION
19       - echo start command
20     #      - $(aws ecr get-login --no-include-email --region $AWS_DEFAULT_REGION)
21   build:
22     commands:
23       - echo Build started on `date`
24       - echo Building the Docker image...
25       - mvn package -Dmaven.test.skip=true
26     #      - docker build -t $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION .
27   post_build:
28     commands:
29       - echo Build completed on `date`
30       - echo Pushing the Docker image...
31     #      - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION
32
33 # cache:
34 #   paths:
35 #     - '/root/.m2/**/*'
```

1. buildspec 에 대한 상세 가이드 -

https://docs.aws.amazon.com/ko_kr/codebuild/latest/using-guide/build-spec-ref.html

2. phases: 필수 시퀀스
3. 빌드의 각 단계 동안 CodeBuild가 실행하는 명령을 표시
4. Install, pre_build, build, post_build 4개의 시퀀스를 적절히 사용할 수 있음

CodeBuild : ECR Connection Policy 설정

The screenshot shows the AWS CodeBuild console. On the left, a sidebar menu is open under '빌드' (Build) with 'CodeBuild' selected. The main area is titled '환경' (Environment). It shows the following details:

- 이미지: aws/codebuild/standard:2.0
- 환경 유형: Linux
- 제한 시간: 1시간 0분
- 서비스 역할: arn:aws:iam::979050235289:role/service-role/codebuild-sample-products-service-role

A red box labeled '1' highlights the '서비스 역할' field. Below this, another red box labeled '2' highlights the JSON editor where the ECR connection policy is being defined.

The JSON editor displays the following policy definition:

```
1 {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Action": [  
6         "ecr:BatchCheckLayerAvailability",  
7         "ecr:CompleteLayerUpload",  
8         "ecr:GetAuthorizationToken",  
9         "ecr:InitiateLayerUpload",  
10        "ecr:PutImage",  
11        "ecr:UploadLayerPart"  
12      ],  
13      "Resource": "*",  
14      "Effect": "Allow"  
15    }  
16  ]  
17 }
```

1. 프로젝트 빌드 - 빌드 선택 - 빌드 세부정보 - 환경 - 서비스 역할 클릭하여 IAM 역할 페이지로 이동
2. 인라인 정책 추가 > json 하여 ECR 관련 정책을 추가
3. 관련 스크립트는 WorkFlowy에 “CodeBuild 와 ECR 연결 정책” 검색
4. 정책명 입력 후, user00-codebuild-policy 검토 및 생성

CodeBuild : Run Pipeline (Docker Build/Push)

The screenshot shows the AWS CodeBuild interface. At the top, there's a file editor titled 'reqres_products / buildspec.yml' with a 'Cancel' button. The code content is as follows:

```
7 phases:
8   install:
9     runtime-versions:
10    java: openjdk8
11    docker: 18
12 pre_build:
13   commands:
14     - echo Logging in to Amazon ECR...
15     - echo $IMAGE_REPO_NAME
16     - echo $AWS_ACCOUNT_ID
17     - echo $AWS_DEFAULT_REGION
18     - echo $CODEBUILD_RESOLVED_SOURCE_VERSION
19     - echo start command
20     - $(aws ecr get-login --no-include-email --region $AWS_DEFAULT_REGION)
21 build:
22   commands:
23     - echo Build started on `date`
24     - echo Building the Docker image...
25     - mvn package -Dmaven.test.skip=true
26     - docker build -t $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION
27 post_build:
28   commands:
29     - echo Build completed on `date` |
30     - echo Pushing the Docker image...
31     - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION
```

Below the editor is a 'Commit changes' dialog with a red border around the 'Commit changes' button. It includes fields for 'Update buildspec.yml' and 'Add an optional extended description...', and buttons for committing directly to the master branch or creating a new branch.

1

1. buildspec.yml 파일의 Docker 관련 주석을 모두 해제 후, 코드 커밋/ 푸쉬
2. 빌드 성공 후 ECR 에 Docker Image 가 정상적으로 생성되었는지 확인

2 ECR에서 파이프라인으로 배포된 이미지 확인

The screenshot shows the AWS ECR 'products' page. A red box highlights the '이미지 (1)' section. It lists one image entry:

이미지 태그	이미지 URI
6dd5866a8971279f62376425d6f374e6fd682c9b	979050235289.dkr.ecr.ap-northeast-1.amazonaws.com/products:6dd5866a8971279f62376425d6f374e

CodeBuild : 빌드 내역 확인

The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with navigation links: '개발자 도구' (Developer Tools), 'CodeBuild' (selected), '소스 • CodeCommit', '아티팩트 • CodeArtifact', '빌드 • CodeBuild' (expanded), '시작하기' (Get Started), '프로젝트 빌드' (Project Builds), '빌드 내역' (Build History) (highlighted with a red dashed box and arrow 1), '보고서 그룹' (Report Groups), '보고서 기록' (Report Logs), '계정 지표' (Account Metrics), '배포 • CodeDeploy', '파이프라인 • CodePipeline', and '설정' (Settings). The main content area shows the build history for a specific build project. The build details are as follows:

상태	시작한 사용자	빌드 ARN	해결된 소스 버전
▶ 진행 중	root	arn:aws:codebuild:ap-northeast-1:979050235289:build/sample-products:8f6dc424-0415-44f0-9cbc-0e2fab8195a8	-
시작 시간	종료 시간	빌드 번호	
3월 19, 2020 3:48 오후 (UTC+9:00)	-	1	

Below the table, there are tabs for '빌드 로그' (Build Log), '단계 세부 정보' (Step Details), '보고서' (Report), '환경 변수' (Environment Variables), and '빌드 세부 정보' (Build Details). A large black box at the bottom displays the first 1000 lines of the build log, with a link to '전체 로그 보기' (View Full Log) and a '테일 로그' (Tail Log) button.

CodeBuild : Cache 적용

The screenshot shows the AWS CodeBuild console with the following steps highlighted:

- 1 S3 서비스에서 Bucket 생성**: A red box highlights the 'Amazon S3' dropdown menu under '캐시 유형' (Cache Type). The selected option is 'Amazon S3'. Below it, a search bar contains the text 'mvn-cache-codebuild'. A red arrow points to the top-left corner of the box.
- 2 파일라인 - Artifact 편집**: A red box highlights the '캐시 경로 접두사 - 선택 사항' (Cache Path Prefix - Optional) input field. A red arrow points to the top-left corner of the box.

Below the main configuration area, there is a note about naming conventions:

캐시 수명 주기 - 선택 사항
경로 접두사에 따라 캐시 버킷 내 전체 또는 일부 객체에 수명 주기 만료 작업을 적용할 수 있습니다.

Buttons at the bottom include '취소' (Cancel), '아티팩트 업데이트' (Artifact Update), and '+ 만료 추가' (Add Expiry).

1. Maven 으로 빌드시,
라이브러리를 캐ши화 시킴
(캐쉬가 없으면 빌드때마다
1~2분 정도 더 소요됨)
2. 캐쉬를 저장할 S3 스토리지
생성
3. 버킷 생성 – 리전을
CodeBuild 생성 리전과 일치
해야함
4. 빌드 선택 – 아티팩트 편집
5. 추가구성 – 캐시 유형을 S3 로
선택 후 버킷 선택
6. buildspec.yml 에서 cache
부분 주석 해제 후 커밋

CodeBuild : VM에서 EKS Connection 설정 (1/4)

1. 쿠버네티스에 접속 하기 위해서는 쿠버네티스 API 주소와 클러스터 어드민 권한이 있는 토큰이 있으면 접속 가능
(실제로 kubectl 명령어가 두개의 조합으로 이루어 져 있음)
2. 쿠버네티스 API 주소 위치

Amazon EKS	Kubernetes 버전	플랫폼 버전
클러스터	1.15	eks.1
Amazon ECR	API 서버 엔드포인트	
리포지토리	https://A4CB98CD955A2E274B343BCEA284621F.yl4.ap-northeast-1.eks.amazonaws.com	

CodeBuild : VM에서 EKS Connection 설정 (2/4)

1. 쿠버네티스 토큰 생성은 ServiceAccount 생성 후 해당 SA 에 cluster-admin 룰을 주면 된다.
2. WorkFlowy 에서 “CodeBuild 와 EKS 연결” 검색 후 토큰 생성

```
→ ~ kubectl -n kube-system describe secret $(kubectl -n kube-system get secret | grep eks-admin | awk '{print $1}')
Name:           eks-admin-token-hffgq
Namespace:      kube-system
Labels:         <none>
Annotations:   kubernetes.io/service-account.name: eks-admin
               kubernetes.io/service-account.uid: 18c3c8c6-a0dc-4f0b-9ba9-a85b6322aa7f
Type:          kubernetes.io/service-account-token

Data
=====
namespace: 11 bytes
token: eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3Nlc3pY2VhY2NvdW50Iiwia3ViZXJuZXRLcy5pb9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOjJrdWJlLXN5c3RlbSIsImt1YmVybml0ZXMuaw8vc2VydmljZWFlY29
```

CodeBuild : VM에서 EKS Connection 설정 (3/4)

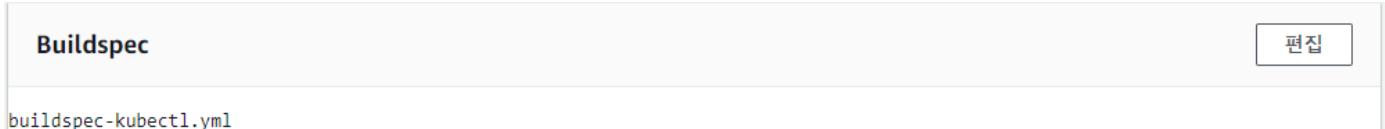
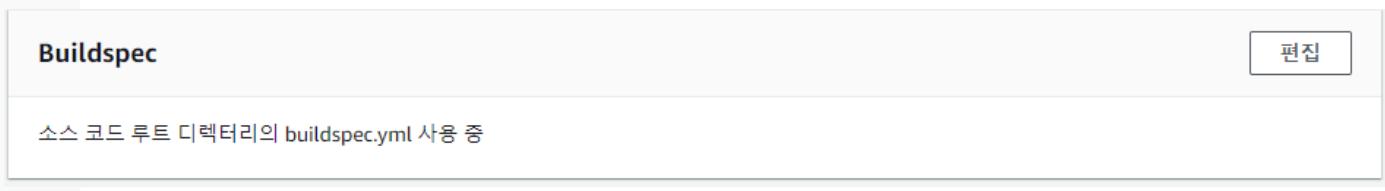
- 빌드 파일의 “환경” 영역 편집 버튼을 눌러, 아래 값을 넣은 후에 저장한다.
- 쿠버네티스 API 주소를 “KUBE_URL” 이라는 이름으로 저장한다.
- 클러스터 어드민 토큰 값을 “KUBE_TOKEN” 이라는 이름으로 저장한다.

환경 변수			
이름	값	유형	제거
AWS_ACCOUNT_ID	979050235***	일반 텍스트	▼ 제거
KUBE_URL	https://A4CB98CD955A.	일반 텍스트	▼ 제거
KUBE_TOKEN	eyJhbGciOiJSUzI1NiIsIm	일반 텍스트	▼ 제거
환경 변수 추가			

CodeBuild : VM에서 EKS Connection 설정 (4/4)

1 파이프라인의 Buildspec을 편집하여, 레파지토리에 있는 buildspec-kubectl.yaml로 수정한다.

2 buildspec-kubectl.yaml 파일을 열어, _PROJECT_NAME: 을 내정보에 맞게 수정하고, 저장한다.



CodeBuild : 파이프라인으로 생성된 결과 확인

- ✓ CodeBuild가 자동 실행 되어,
EKS 클러스터에 배포 후
서비스가 정상적으로 기동
되었는지 확인한다.
- ✓ Kubectl get all

```
root@labs--1147635527:/home/project# kubectl get all
NAME                                         READY   STATUS    RESTARTS   AGE
pod/user30-products-5f4d77645f-2x86p        1/1     Running   0          45m

NAME                  TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)      AGE
service/kubernetes   ClusterIP   10.100.0.1    <none>       443/TCP     5h15m
service/user30-products   ClusterIP   10.100.202.42  <none>       8080/TCP    45m

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/user30-products  1/1     1           1          45m

NAME                           DESIRED  CURRENT  READY   AGE
replicaset.apps/user30-products-5f4d77645f  1        1        1      45m
root@labs--1147635527:/home/project#
```

AWS CodeBuild를 활용한 CI/CD 2nd Lab.

From Page.38

1. Github에 있는 소스 fork

https://github.com/event-storming/reqres_delivery

2. Docker image를 저장하기 위한 공간인 ECR에 Repository 생성

3. reqres_products 레파지토리에서 buildspec-kubectl.yml 파일 내용 복사해 오기

4. Fork 받은 리파지토리의 Root에서 buildspec.yml을 편집하여 5행의 이미지명 수정
- 위에서 생성한 ECR Repository와 동일한 이름 입력



CodeBuild Webhook Event Filter (1/2)

- 환경 별로 다른 빌드를 하고 싶으면 빌드 파일을 다르게 생성하여 빌드 프로젝트를 각자 생성하면 된다.
(예 : 개발 환경 - buildspec-dev.yml, 운영환경 - buildspec-prod.yml)
- 기존 프로젝트의 webhook 트리거를 잠시 제거하고, buildspec-sample.yml 파일로 빌드 프로젝트를 생성한다.
- 특정 파일 수정일때 트리거 작동 안하도록 설정

- Webhook 이벤트 항목을 편집한다.
- 이러한 조건에서 빌드 시작 안함에서 FILE-PATH 부분에 “^buildspec.*” 값을 넣는다.
- buildspec 으로 시작하는 모든 파일일때 웹훅 이벤트가 동작 안한다.
- 정규식으로 작성해야 한다.
- buildspec-sample.yml 파일을 수정하여 트리거가 작동하는지 확인 한다 (작동하면 안됨)

Webhook 이벤트 필터 그룹 1

이벤트 유형

PUSH X

▼ 이러한 조건에서 빌드 시작

ACTOR_ID - 선택 사항	HEAD_REF - 선택 사항	BASE_REF - 선택 사항	FILE_PATH - 선택 사항
------------------	------------------	------------------	-------------------

▼ 이러한 조건에서 빌드 시작 안 함

ACTOR_ID - 선택 사항	HEAD_REF - 선택 사항	BASE_REF - 선택 사항	FILE_PATH - 선택 사항
------------------	------------------	------------------	-------------------

^buildspec.*

CodeBuild Webhook Event Filter (2/2)

1. 태그 이벤트일때 트리거 작동하도록 설정

- 이러한 조건에서 빌드 시작 항목 – HEAD_REF 에 “`^refs/tags/.*`” 를 입력한다.
 - > 팁 : (아래 이미지처럼 빌드 시작안함도 같이 있을 경우, buildspec 파일만 변경하고, tag 를 하여도 트리거는 동작하지 않는다.)
 - > 팁 : 필터를 여러개 생성해도 1개만 만족하면 트리거는 작동한다
- 저장 후, README.md 파일을 수정하여 트리거가 작동하는지 확인한다. (빌드 시작 안함)
- github Tag 를 설정 한 후 트리거가 작동하는지 확인 한다. (빌드 시작함)
- HEAD_REF 에 “`^refs/heads/dev$`” 를 추가하여 dev 브랜치에서만 작동하는것을 연습해보자.

Webhook 이벤트 필터 그룹 1

이벤트 유형

PUSH X

▼ 이러한 조건에서 빌드 시작

ACTOR_ID - 선택 사항	HEAD_REF - 선택 사항	BASE_REF - 선택 사항	FILE_PATH - 선택 사항
<input type="text"/>	<input type="text" value="^refs/tags/.*"/>	<input type="text"/>	<input type="text"/>

▼ 이러한 조건에서 빌드 시작 안 함

ACTOR_ID - 선택 사항	HEAD_REF - 선택 사항	BASE_REF - 선택 사항	FILE_PATH - 선택 사항
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text" value="^buildspec.*"/>

```
86 [Container] 2020/03/20
87 tag name2 : tag/v1.0.1
88
89 [Container] 2020/03/20
90
91 [Container] 2020/03/20
92 tag name3 : v1.0.1
93
```

“ Appendix: Contract Test

- Deployment Strategies
- Contract Test

배포 전략별 비교

When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.



Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■ ■ ■	■ ■ ■	■ ■ ■	□ □ □
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■ ■ ■	■ ■ ■	■ ■ □	■ ■ □
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■ ■ ■	□ □ □	■ ■ □	■ ■ □
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■ ■ ■	■ □ □	■ ■ □	■ ■ □
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■ ■ ■	■ □ □	■ ■ □	■ ■ ■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■ ■ ■	□ □ □	□ □ □	■ ■ ■

Consumer Driven Testing (Contract Test)

- **What** : 서비스 제공자와 사용자간 프로토콜, API 스펙, kind of Component Test
- **Why** : 서비스 제공자가 내 서비스를 사용하는 소비자에 대한 정보 및 규약 유지

Contract Testing

for Applications with Microservices



Ex. Spring Cloud Contract, Pact

- 소비자가 Contract를 제공자에 Groovy로 작성
- 서비스 빌드 시, Spring Cloud Contract는 자동으로 계약서를 테스트 코드로 Generarte.
- 생성된 테스트 코드를 Unit Test와 같이 테스트
- 테스트 실패 시, 서비스 코드 배포 차단

“배포되어 실제 사용자가 사용중인 API는 항상 정상적으로 유지되도록 하고싶다.”

(하위호환성, Backward Compatibility)

MSA 테스트 자동화

1. 단위 테스트 : 소프트웨어 구성 요소를 개별적으로 테스트

> Unit Test : 응용 프로그램이 제대로 작동하고 있다고 확인할수 있는 테스트. 보통 Unit 테스트는 소스코드를 빌드 하는 단계에서 같이 실행

> Regression(회귀) Test : 개선 사항 및 버그 수정으로 인해 발생할 수있는 버그를 발견하는 테스트. 운영중에 버그가 발생했을 경우 이를 수정하면서 해당 버그를 발견할 수 있는 테스트 케이스를 만들고 이를 테스트 자동화에 적용하는 방법

2. 통합 테스트 : 소프트웨어 구성 요소를 함께 테스트

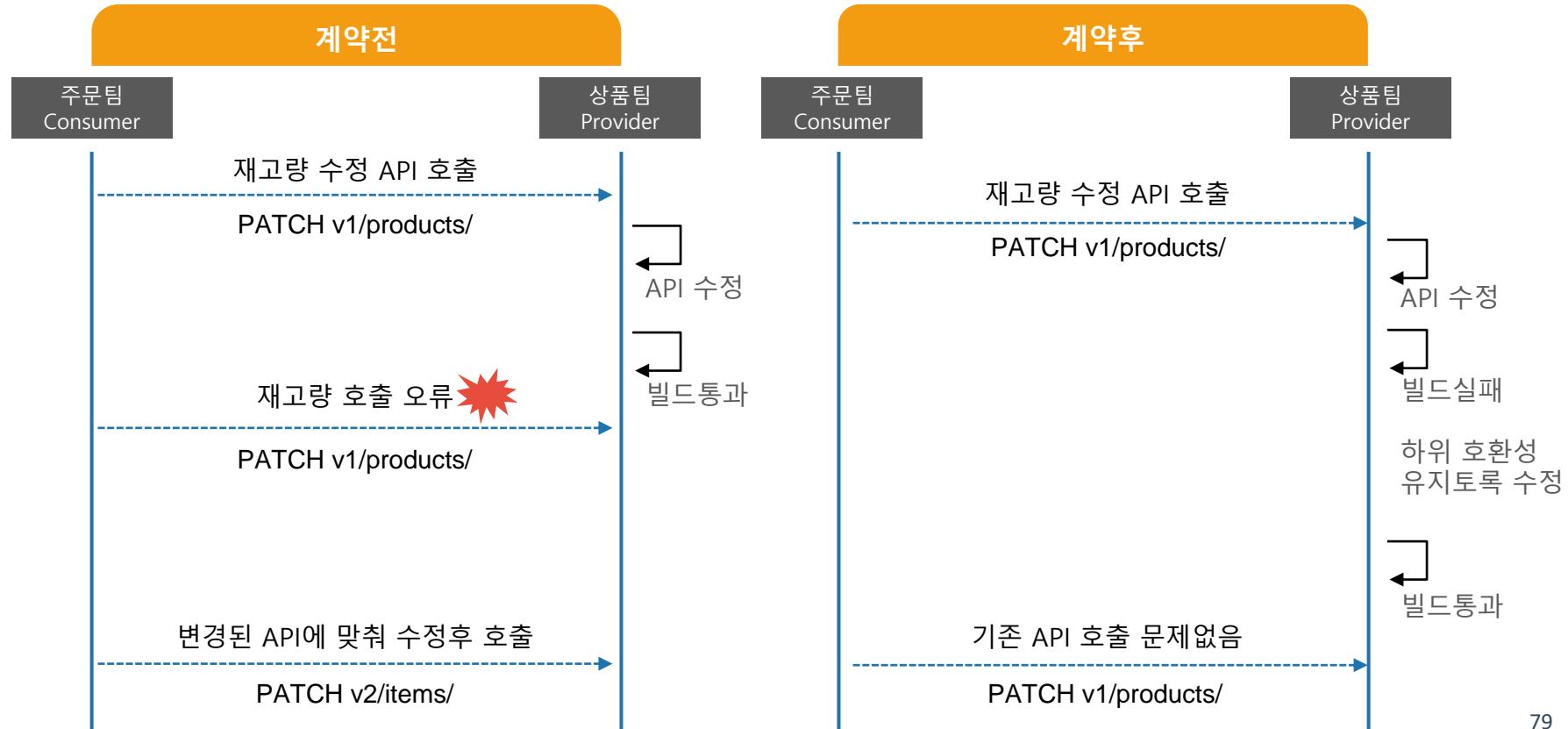
> API Test : 개발된 API 가 정상 작동하는지 테스트. 테스트를 위한 서비스를 생성 후, 테스트 후, 서비스 제거 방식으로 실행

> Contract Test : 다른 서비스와 계약서를 만든 후에 계약서를 유지시키는 테스트. 소스코드를 빌드 하는 단계에서 같이 실행

3. E2E 테스트 : 모든 소프트웨어 구성 요소가 예상대로 작동하는지 확인

> 테스트 자동화 힘듬: 모든 서비스를 올려야 하고, 브라우저 테스트나 클릭 테스트등 유저가 직접 테스트를 해야하는 경우가 많아서 비용이 높음. CasperJS, Testcafe 등 별도의 툴을 사용하여 자동화 하여야 함

Contract Test



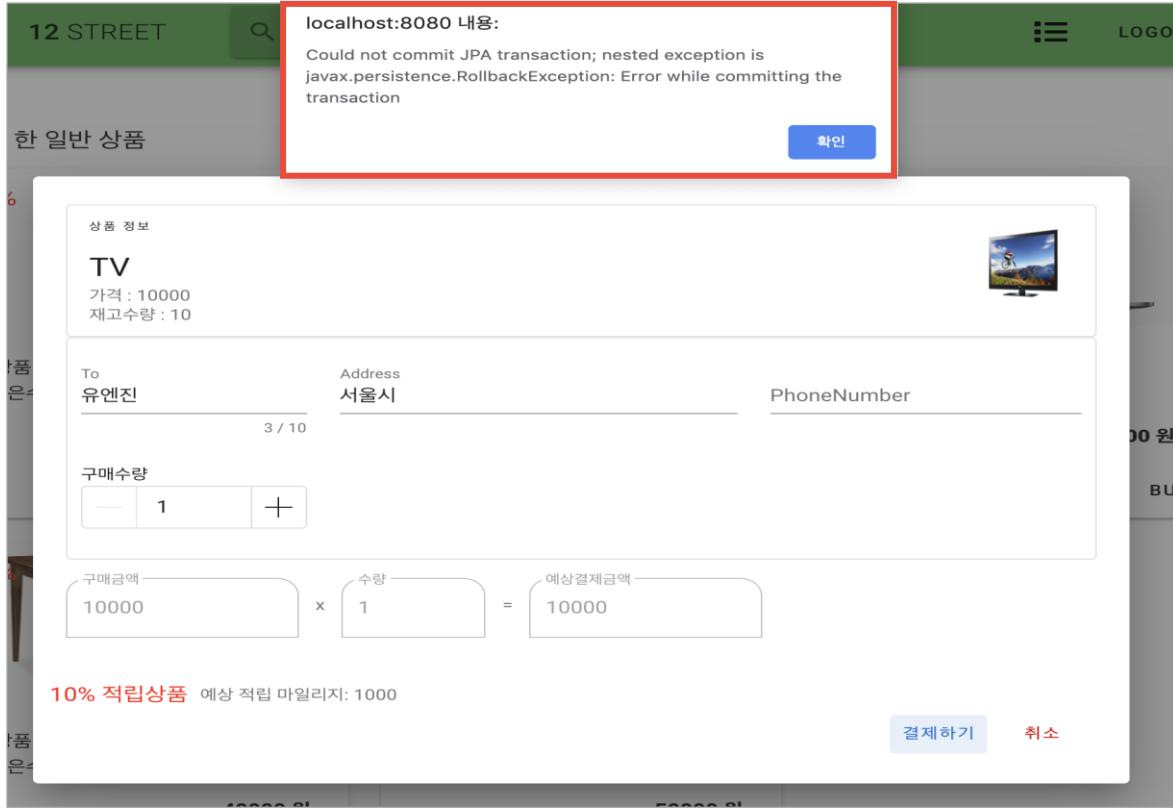
상품팀에서 API를 일방적으로 변경

소스 위치

products / src / main / java / com / example / template / ProductController.java

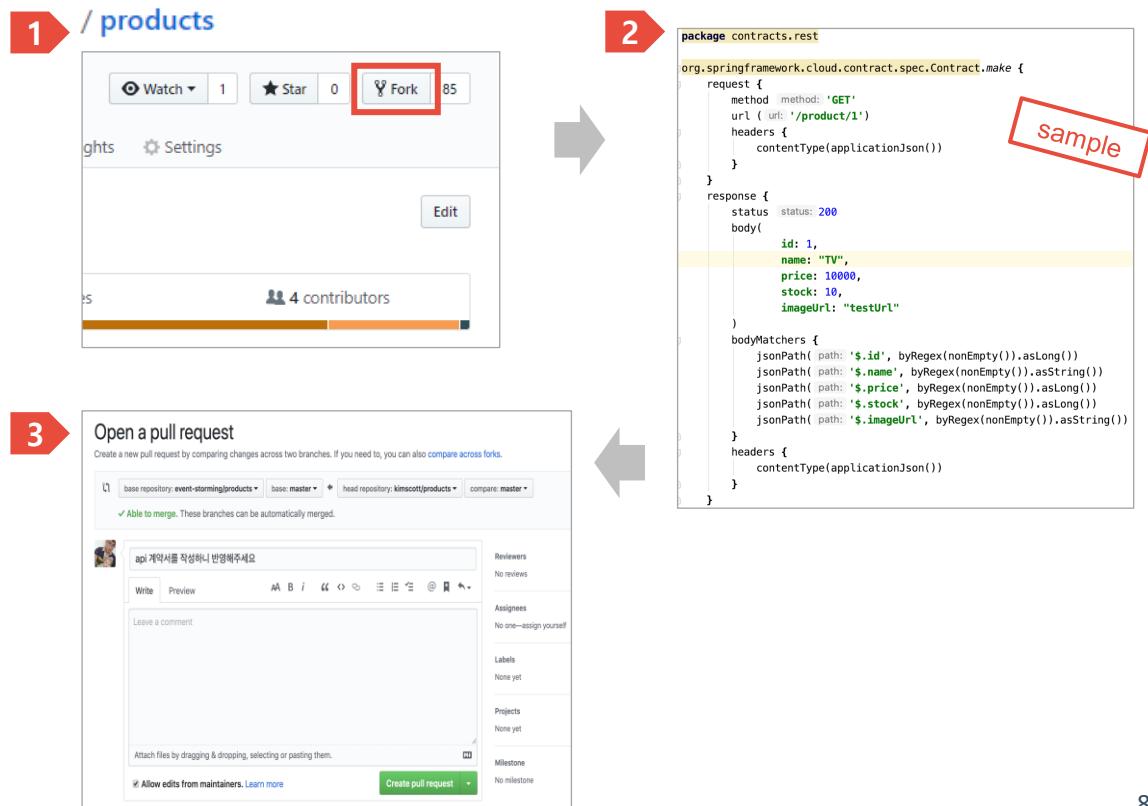
```
13  
14     @GetMapping("/product/{productId}") /item/{productId} 로 변경  
15     Product productStockCheck(@PathVariable(value = "productId") Long productId) {  
16         return this.productService.getProductById(productId);  
17     }  
18 }  
19 }
```

주문팀의 서비스 장애 발생



계약체결(1/2) - 주문팀에서 계약서 작성 후, 상품팀에 체결 요청

1. 상품팀 소스 복사 (포크 생성)
2. 주문팀이 계약서 생성
3. 주문팀에서 생성한 계약서 (productGet.groovy) 파일을 상품팀에 Pull request 요청함



계약체결(1/2) - 주문팀에서 계약서 작성 후, 상품팀에 체결 요청

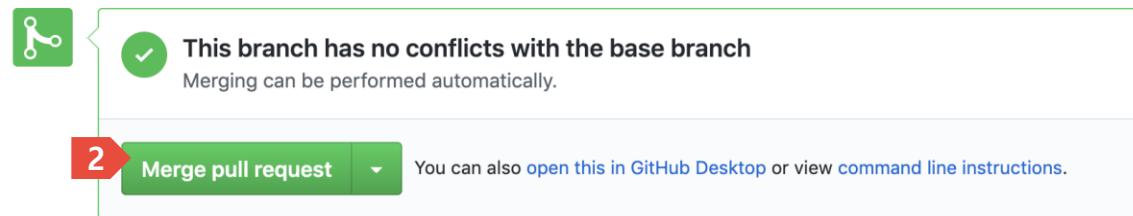
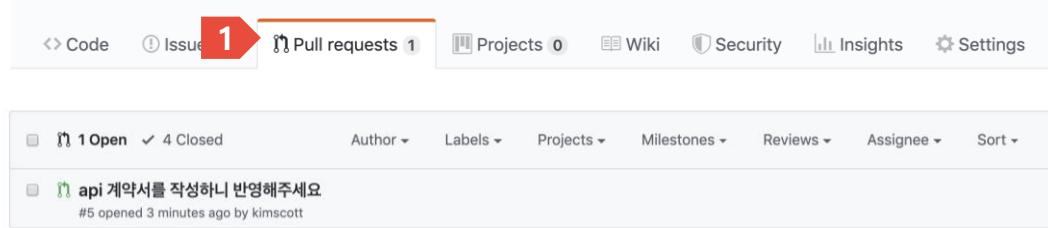
```
package contracts.rest

org.springframework.cloud.contract.spec.Contract.make {
    request {
        method method: 'GET'
        url ( url: '/product/1')
        headers {
            contentType(applicationJson())
        }
    }
    response {
        status status: 200
        body(
            id: 1,
            name: "TV",
            price: 10000,
            stock: 10,
            imageUrl: "testUrl"
        )
        bodyMatchers {
            jsonPath( path: '$.id', byRegex(nonEmpty()).asLong())
            jsonPath( path: '$.name', byRegex(nonEmpty()).asString())
            jsonPath( path: '$.price', byRegex(nonEmpty()).asLong())
            jsonPath( path: '$.stock', byRegex(nonEmpty()).asLong())
            jsonPath( path: '$.imageUrl', byRegex(nonEmpty()).asString())
        }
        headers {
            contentType(applicationJson())
        }
    }
}
```

sample

계약체결(2/2) - 상품팀에서 계약서 수락

- 상품팀 : Pull Request 메뉴 선택
- 상품팀은 해당 계약서를 accept 하여 반영함
- 상품 서비스는 이제부터는 계약서를 안지켰을때 아예 배포가 안됨



계약 체결

1. 주문팀에서 계약서 내용 복사
2. 상품팀에 계약서 붙여넣기
3. Commit

The image shows two screenshots of a GitHub interface. The top screenshot displays a commit page for a file named 'orders / productGet.groovy'. The commit message is 'change 계약서 파일' (Change contract file) by 'kimscott'. It shows 1 contributor and 31 lines (30 sloc) of code. The code snippet starts with 'package contracts.rest;'. A red arrow labeled '1' points to the file path in the commit message. The bottom screenshot shows a code editor with a file path 'products / src / test / resources / contracts / rest / productGet.groovy' highlighted in blue. The editor has tabs for 'Edit new file' and 'Preview', and settings for 'Spaces', '2', and 'No wrap'. A red arrow labeled '2' points to the file path in the code editor.

계약체결 후, 상품팀은 계약 위반으로 배포 실패함

Azure-pipeline에서 mvn package 단계 실패

The screenshot shows the 'Errors' section of an Azure Pipeline run. It contains one entry under 'Maven': 'Build failed.' Below this is a table titled 'Jobs' with one row. The row has columns for 'Name' (Job), 'Status' (Failed), and 'Duration' (47s). The 'Job' column is highlighted with a red border.

Name	Status	Duration
Job	Failed	47s



The screenshot shows the 'Jobs in run #2020020...' page. It lists various jobs: Initialize job (Passed, 2s), Checkout kimscott/produc... (Passed, 3s), Cache Maven local repo (Passed, 10s), Maven (Failed, 29s), CopyFiles (Passed, <1s), PublishBuildArtifacts (Passed, <1s), Docker (Passed, <1s), Post-job: Cache Maven (Passed, <1s), Post-job: Checkout kimscott/produc... (Passed, <1s), and Finalize Job (Passed, <1s). The 'Maven' job is expanded to show its details.

Maven

```
[INFO] [ERROR] Failures:
[ERROR] RestTest.validate_productGet:33
Expecting:
<404>
to be equal to:
<200>
but was not.
[INFO]
[ERROR] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0
[INFO]
[INFO] BUILD FAILURE
[INFO]
[INFO] Total time: 25.235 s
[INFO] Finished at: 2020-02-03T08:06:52Z
[INFO]
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.22.1:test (defa
[ERROR]
[ERROR] Please refer to /home/vsts/work/1/s/target/surefire-reports for the individual test resu
[ERROR] Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date]
[ERROR] -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the following
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
[ERROR] The process '/usr/share/apache-maven-3.6.3/bin/mvn' failed with exit code 1
[ERROR] Could not retrieve code analysis results - Maven run failed.
```

상품팀에서는 하위 호환성을 유지하며, 추가 API 제공

products / src / main / java / com / example / template / ProductController.java

```
@GetMapping("/item/{productId}")
Product productStockCheck(@PathVariable(value = "productId") Long productId) {
    return this.productService.getProductById(productId);
}

@GetMapping("/product/{productId}")
Product productStockCheck1(@PathVariable(value = "productId") Long productId) {
    return this.productService.getProductById(productId);
}
```

기존의 하위
호환성 API 유지