# CI/CD with AWS DevOps Pipeline



uEngine

# What is CI/CD? DevOps? Pipeline?

- CI/CD 라는 용어는 애자일 소프트웨어 개발 방법론 에서 나옴
- 애자일은 요구사항을 스프린트에 마추어 작게 , 유연하게 개발
- DevOps의 핵심은 개발자와 테스터, 고객이 모든 단계에 참여


- CI / CD는 민첩한 개발을 구현하기 위해 올바른 자동 테스트 도구를 사용하는 DevOps 전략
- CI/CD 를 하기 위해서는 자동화된 도구가 필요 - 이러한 도구들의 묶음을 CI/CD 파이프라인 이라고함
- 즉! 어플리케이션의 통합 및 테스트 단계에서부터 제공 및 배포에 이르는 어플리케이션의 라이프사이클 전체에 걸쳐 자동화와 모니터링을 제공하는 프로세스들의 묶음을 의미
- CI / CD, Agile 및 DevOps 는 같은 목표를 가지고 있음 - **짧은 시간에 더 나은 소프트웨어를 만들기!!**

# Continuous Integration

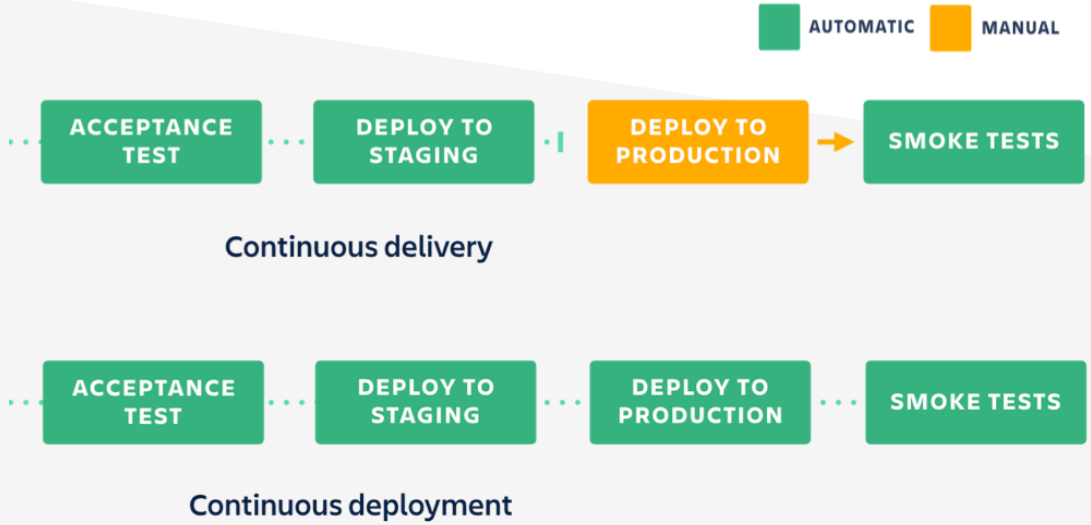
- 개발자를 위한 자동화 프로세스인 지속적인 통합(Continuous Integration)을 의미
- **개발코드를 통합할 때의 문제점을 해결하고,**
  **자동화시켜 지속적으로 유지시키는 방법**
- 코드를 커밋만 치면 자동으로 빌드, 통합을 하고, 테스트를 하는 과정을 의미

- 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.
  - 테스트에 시간이 오래 걸리면 안됩니다.
  - 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**

- 성공적인 CD 를 하려면?
  - 개발/스테이징/운영 환경에 동일한 배포 프로로세스를 적용
  - 모든 환경에 동일한 패키지를 사용 (환경별 구성과 패키지를 다르게 유지해야 한다는 것을 의미)
  - 빠른 롤백이 가능하도록 구성
  - 모니터링 할 수 있는 도구 필요

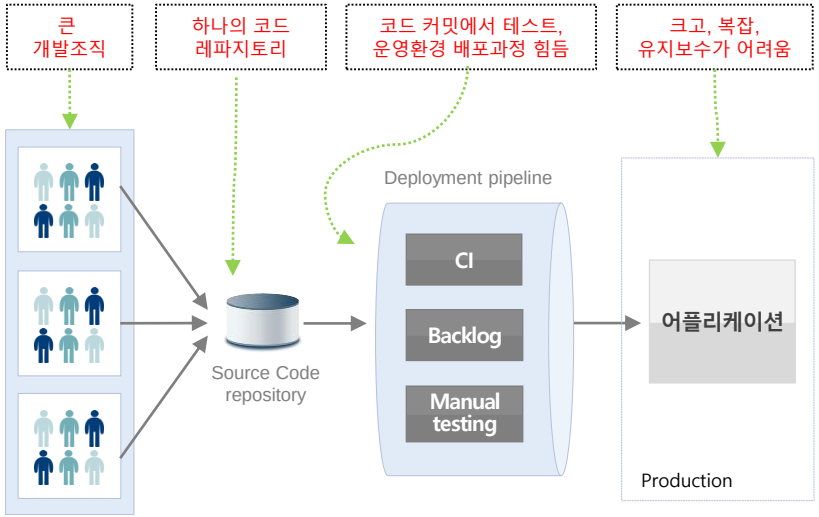


# Table of content

# Process Change : 열차말고 택시를 타라!

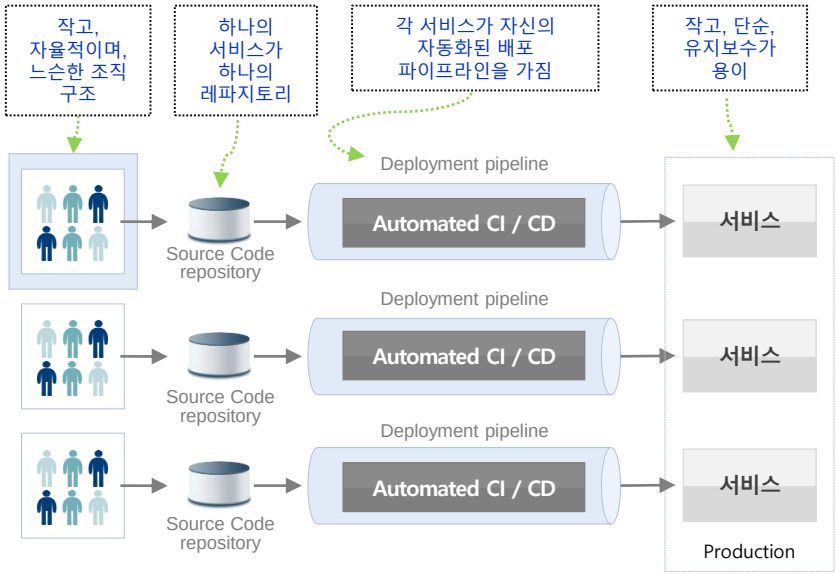
## 모노리식 개발 및 운영환경

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




## 마이크로서비스 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




Modified from Microservices Patterns, Chris Richardson, Manning, 2018

# Test Change : 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

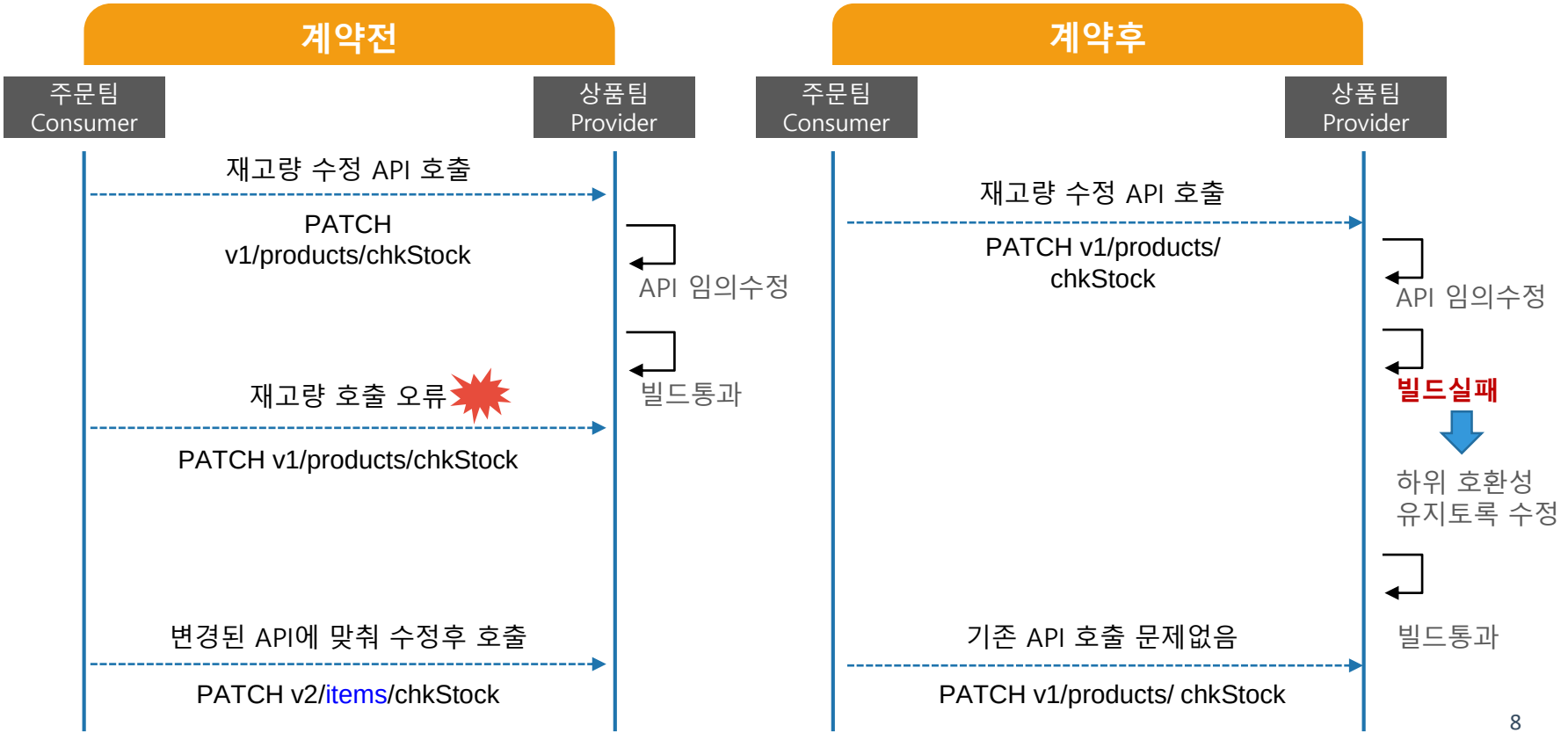

Ex. Spring Cloud Contract, Pact

- 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)

# Contract Test



# Continuous Delivery

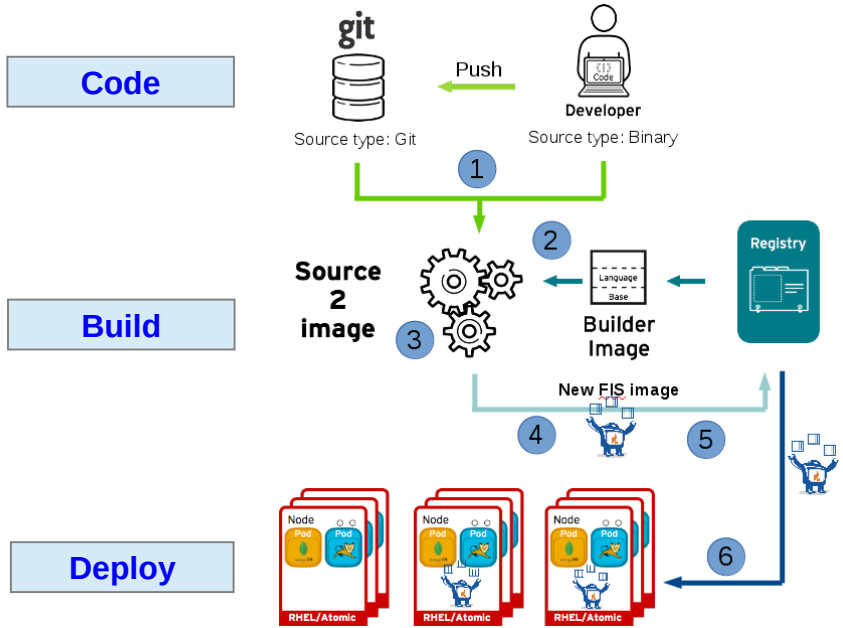
**Amazon, Google, Netflix, Facebook, Twitter**는 얼마나 자주 배포할까요?

| Company | Deploy Frequency | Deploy Lead Time | Reliability | Customer Responsiveness |
|---|---|---|---|---|
| Amazon | 23,000 / day | Minutes | High | High |
| Google | 5,500 / day | Minutes | High | High |
| Netflix | 500 / day | Minutes | High | High |
| Facebook | 1 / day | Hours | High | High |
| Twitter | 3 / week | Hours | High | High |
| Typical enterprise | Once every 9 months | Months or quarters | Low / Medium | Low / Medium |

출처: 도서 The Phoenix Project

# DevOps toolchain

자동화 도구 필요




GitLab

Travis CI

docker

Maven

kubernetes

DC/OS

# DevOps platform

컨테이너화 필요

| Container | Workload Distribution Engine (Container Orchestrator) | PaaS |
|---|---|---|
| • Docker | • Kubernetes | • Google Cloud Platform |
| | • Docker SWARM(toy) | • Redhat Open Shift |
| | • Mesos Marathon(DCOS) | • Amazon EKS |
| | | • MS Azure |
| • Warden(Garden) | • Cloud Foundry | • Heroku |
| | | • GE's Predix |
| | | • Pivotal Web Services |
| • Hypervisor | • CF version 1 | • Amazon Beanstalk |
| | • Engine yard…. | |

# 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.

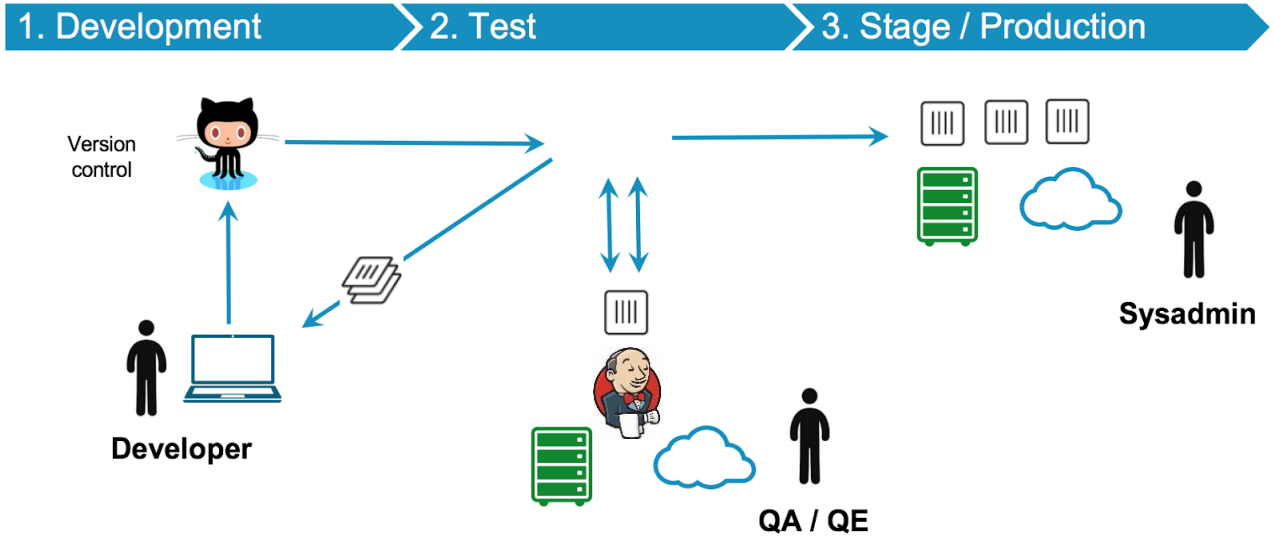
If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.

| 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 | ✓ | ✓ | ✗ | ■■■ | □□□ | □□□ | ■■■ |

Container Solutions

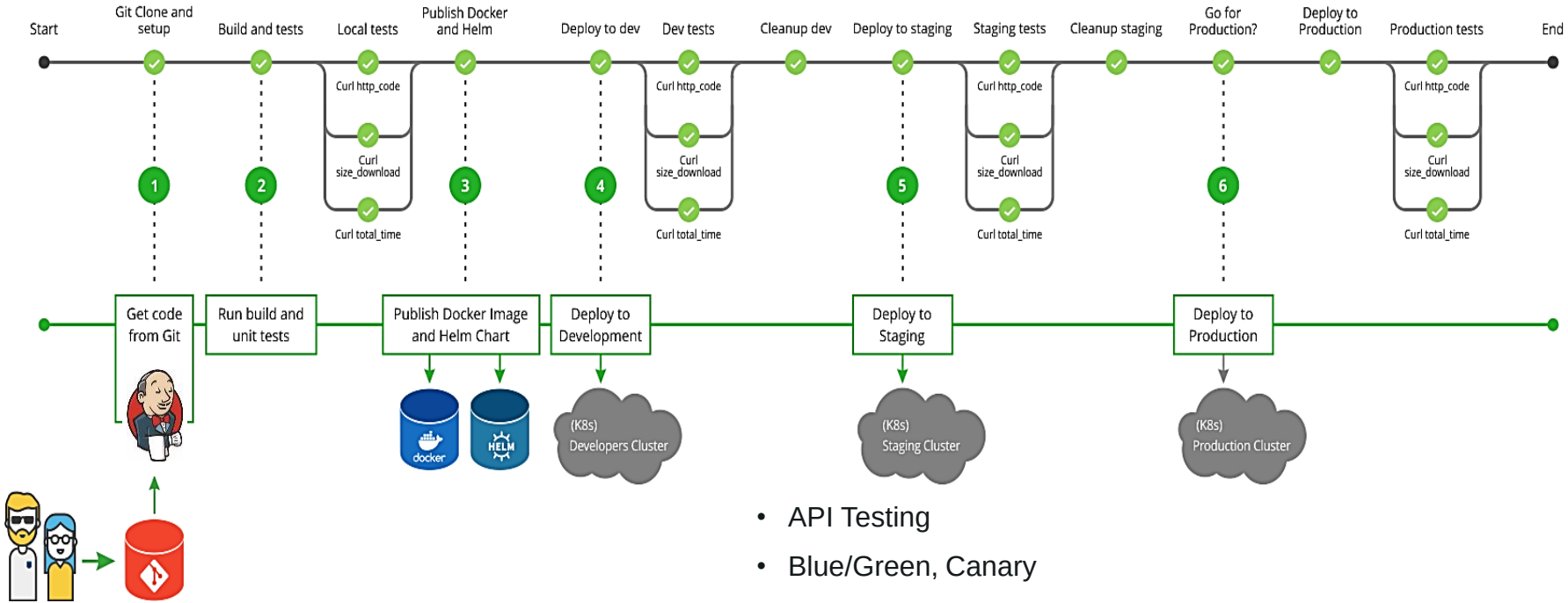
# CI/CD Tools – conventional CI/CD

## CI /CD Workflow



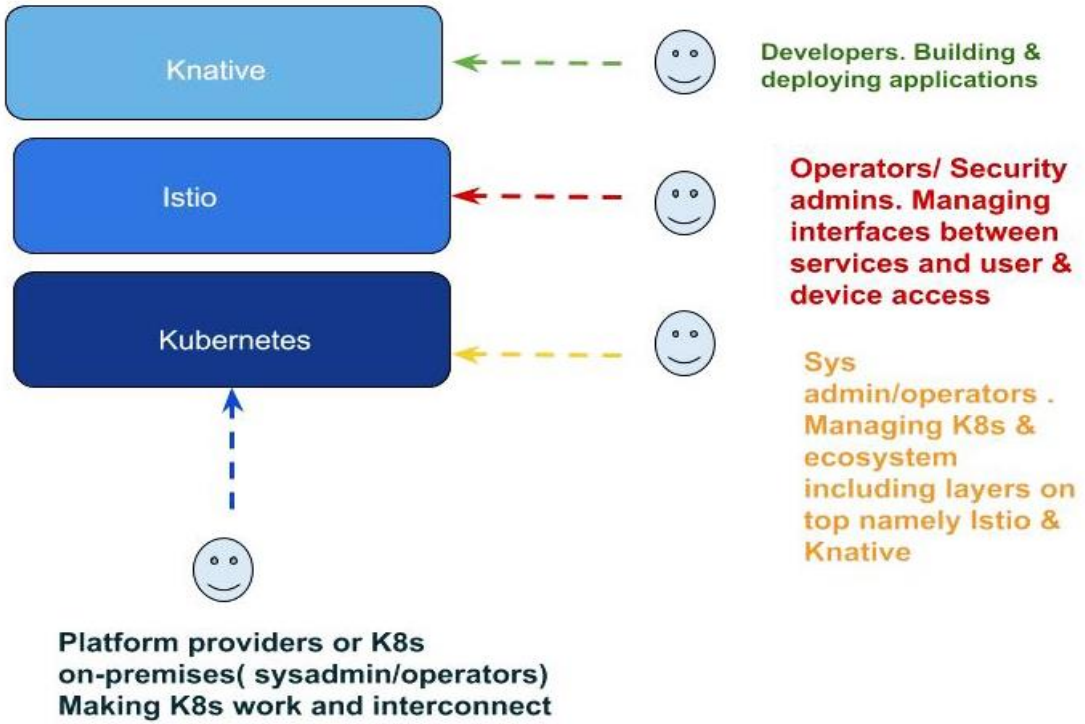

- 잦은 배포
- 일반 서버에 배포
- 테스트 자동화/커버리지
- 도구: Jenkins, Travis, Gitlab 등

# CI/CD Tools – Container-based




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

# CI/CD Tools - Serverless




- Infrastructure As A Code
- Application Configuration 과 Infra Configuration을 하나의 설정에 통합
- 단일 도구

# Table of content

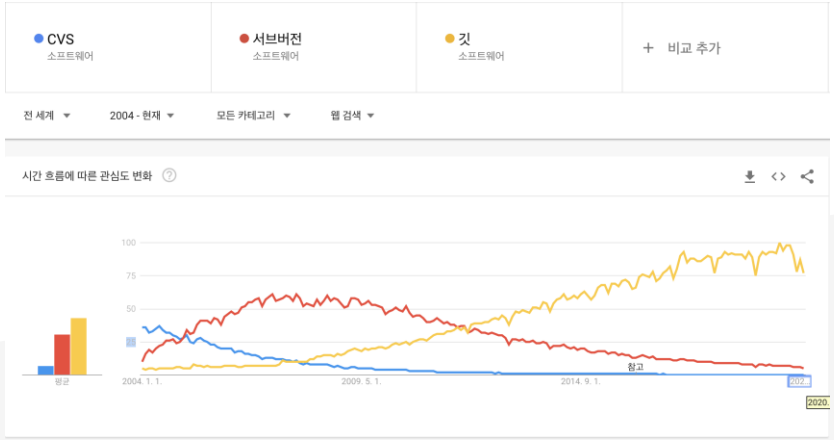
# 형상관리 - History

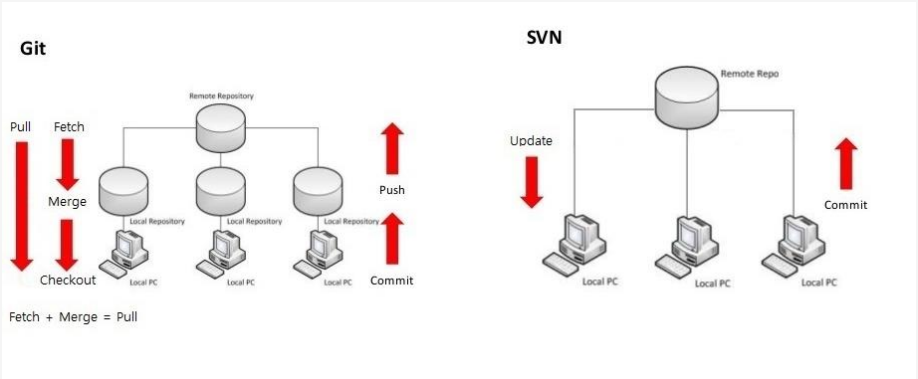
- **형상관리란? :**
  **소스의 변화를 끊임없이 추적하고, 버전별로 관리**

- 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

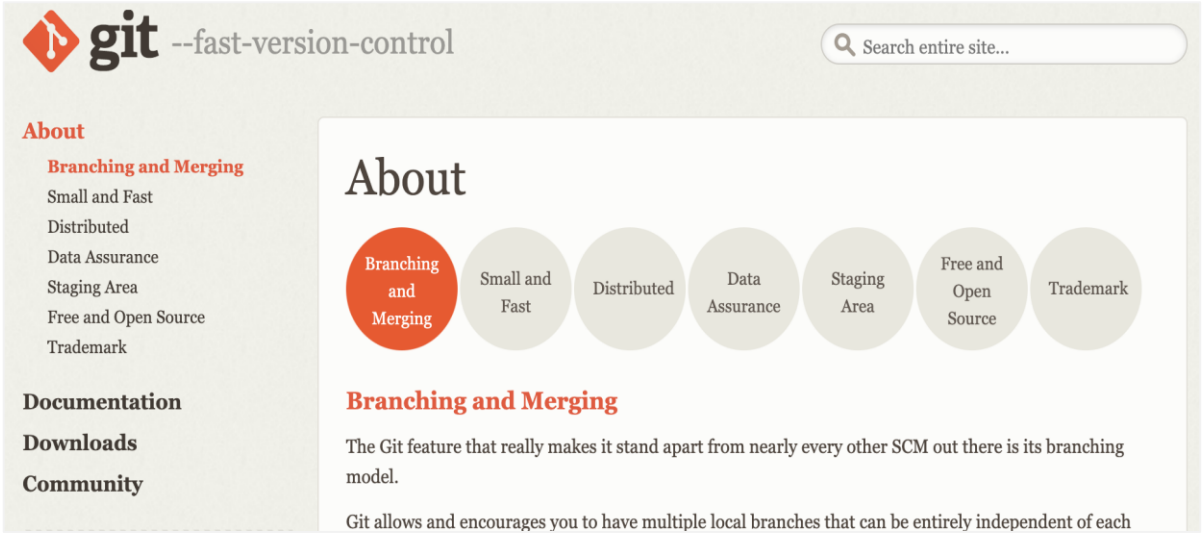
- GIT
  - 2005년 리누스 토발즈에 의해 시작
  - 매우 빠른 속도와 분산형 저장소. SVN보다 많은 기능을 지원
  - 개발자가 자신만의 commit history를 가질 수 있음
  - 저장소 분리로, 복원이 용이



# 형상 관리 – Git

1. Branch and Merge : 새로운 기능 패치시 브랜치를 생성하고 다시 메인코드로 병합가능
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

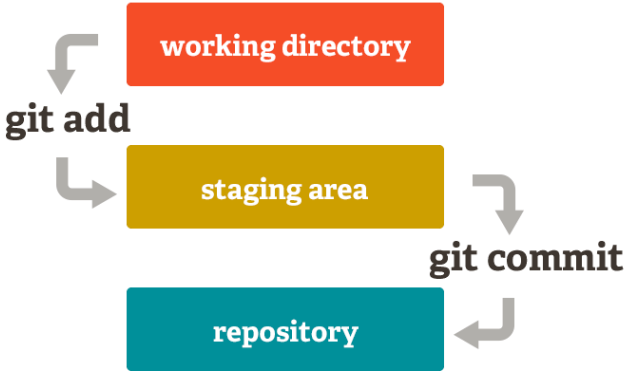


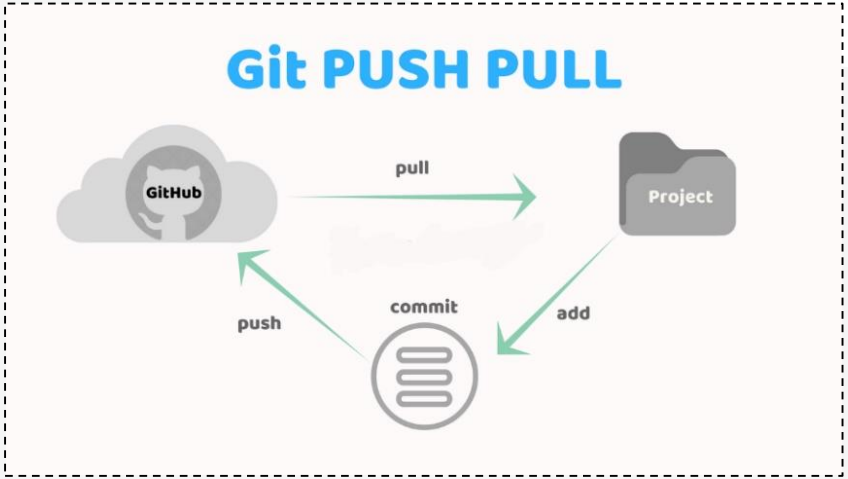

https://git-scm.com/about/

# 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
- git remote
- git clone
- git pull
- git push



# Table of content

# Java build automation tools

- Ant , Maven, Gradle 비교
- Maven 사용법

- https://maven.apache.org/
- https://www.baeldung.com/ant-maven-gradle
- https://sjh836.tistory.com/131
- https://www.slideshare.net/sunnykwak90/ss-43767933?qid=edd5470b-614e-4e16-bbd1-77484fe674ac&v=&b=&from_search=6

# 빌드 자동화 툴

- 빌드 자동화란? : 자바 소스를 Compile하고 Package해서 Deploy하는 일을 자동화 해주는 것

- Apache Ant
  - Java Build Tool, Another Neat Tool
  - 2000 년 출시
  - Base build file : build.xml
  - 유연함이 장점이지만 (모든 명령을 직접작성) , 규칙이 없기때문에 유지보수가 어려움
- Apache Maven
  - Java Build를 위한 Ant 의 불편함을 해소하고자 2004년 출시
  - 규칙을 정하고 Goals 라는 사전 정의된 command 를 제공
  - Base build file : pom.xml
- Gradle - 다양한 언어 지원
  - Ant 와 Maven 의 장점을 모아 2012년 출시
  - Android OS 의 빌드 도구로 채택
  - 프로그래밍 언어 형식으로 유연함이 장점 ( Groovy 파일로 작성)
  - Base build file : build.gradle

# Maven 핵심 개념

- 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 포함하고있다.  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>
```

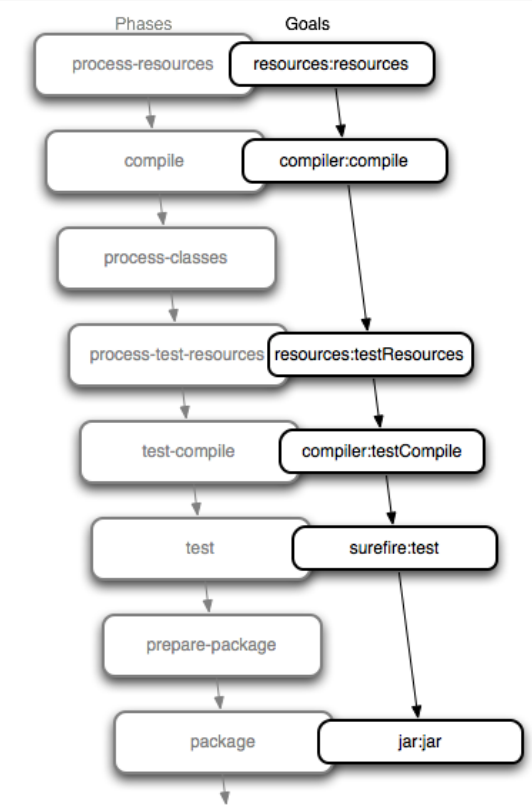
# Lifecycle

Phase | Plug-in | Goal

- 메이븐 프로젝트 생성에 필요한 단계들 (phases) 의 묶음

- clean, default, site 세가지로 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** : target디렉토리 하위에 jar, war, ear등 패키지파일을 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>
  <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>
```

  - 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 )

# 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

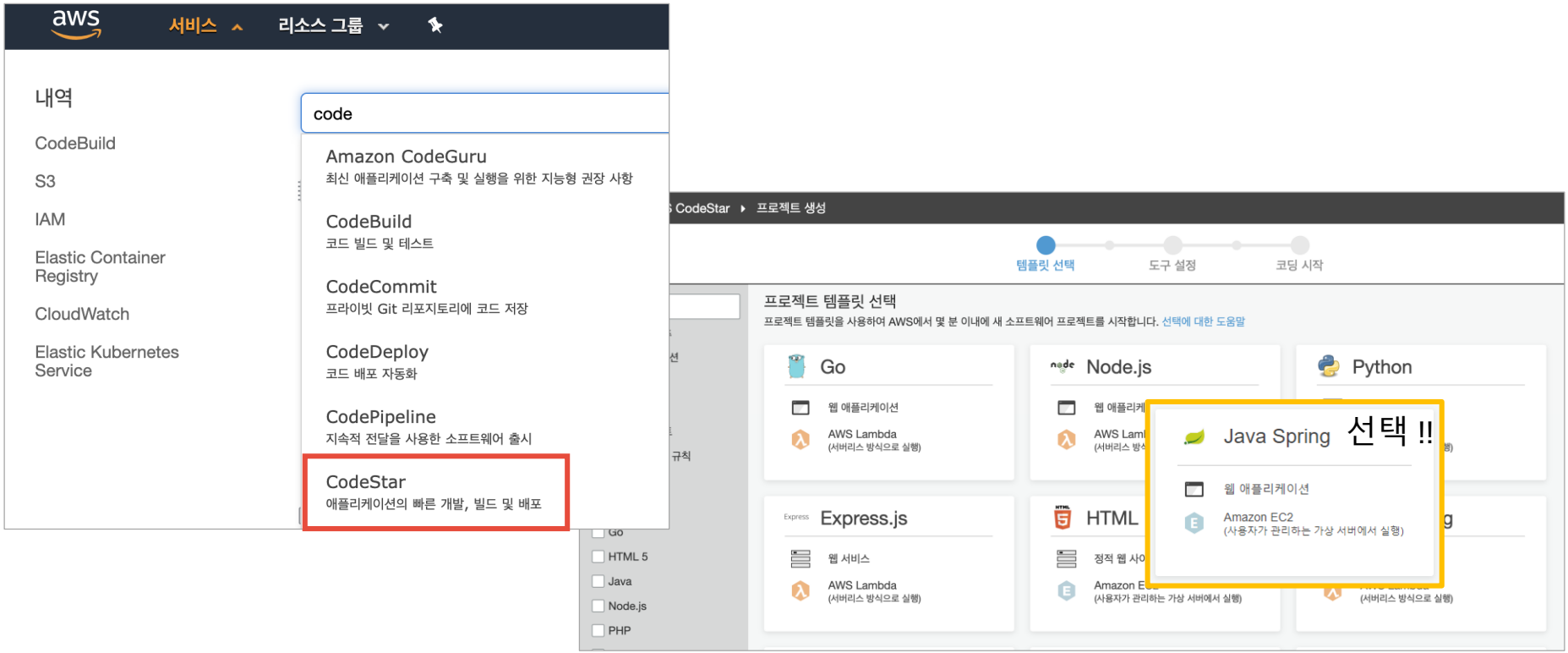
- POM은 프로젝트 객체 모델(Project Object Model)
- 프로젝트 당 1개
- 프로젝트의 root에 존재
- <groupId> , <artifactId>, <version> 으로 자원을 식별

- <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

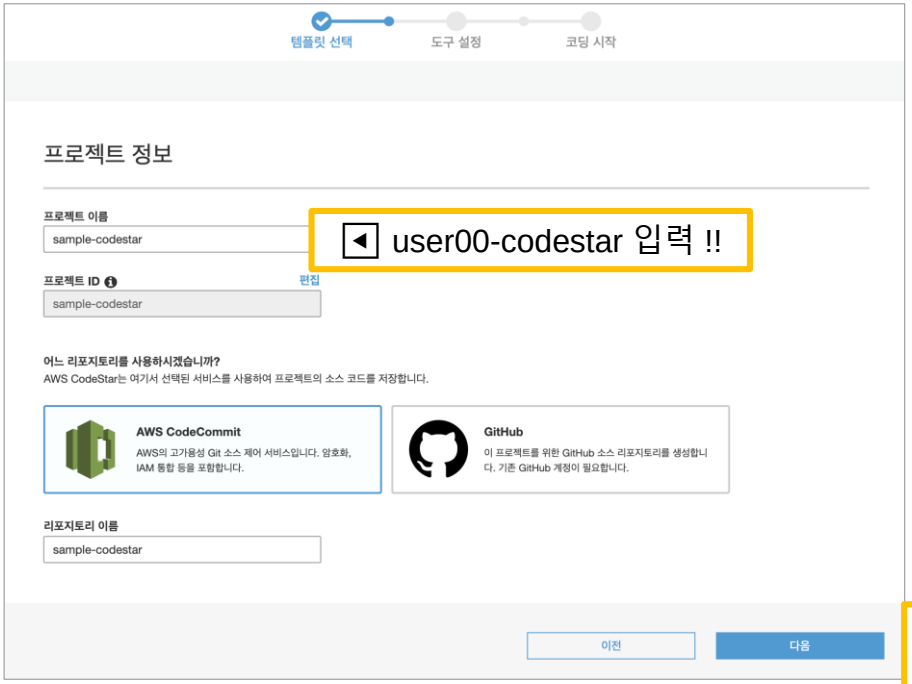
# AWS DevOps – CodeStar

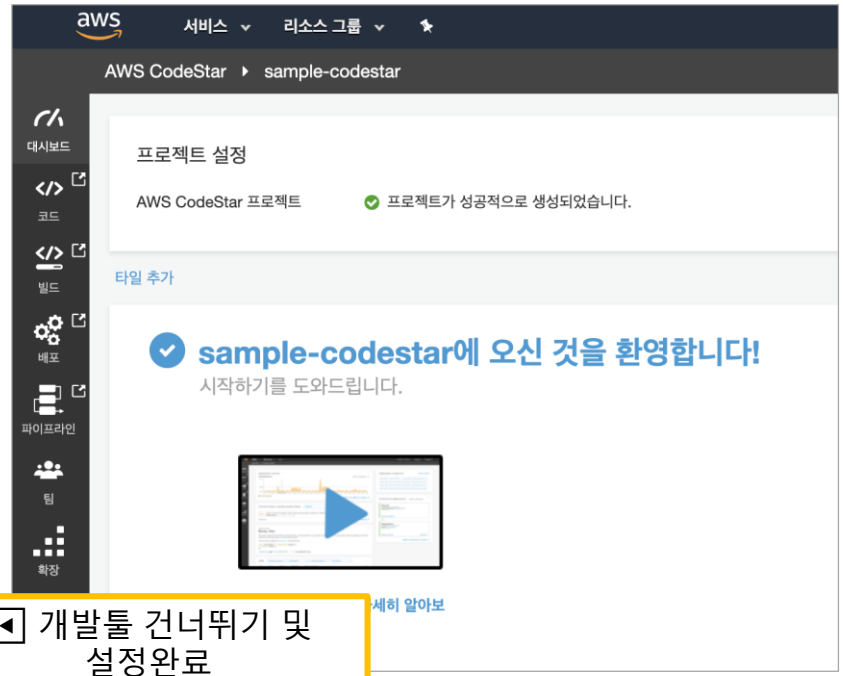
- 웹 어플리케이션 템플릿을 사용, AWS에서 애플리케이션을 빠르게 개발, 빌드, 배포할 수 있는 통합 도구



# Aws DevOps - CodeStar

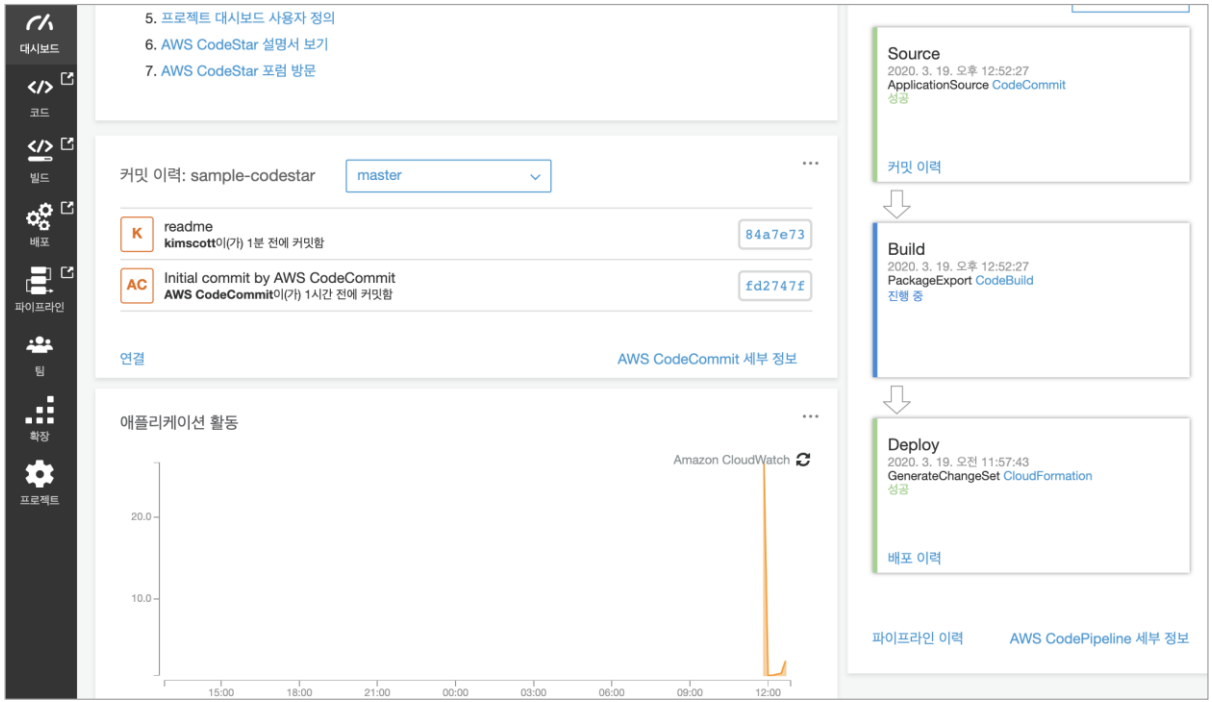
- 웹 어플리케이션 템플릿을 사용, AWS에서 애플리케이션을 빠르게 개발, 빌드, 배포할 수 있는 통합 도구



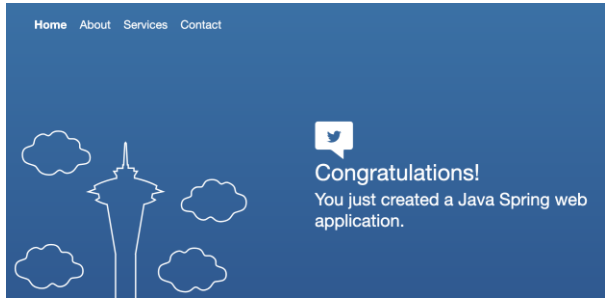



# Aws DevOps - CodeStar

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

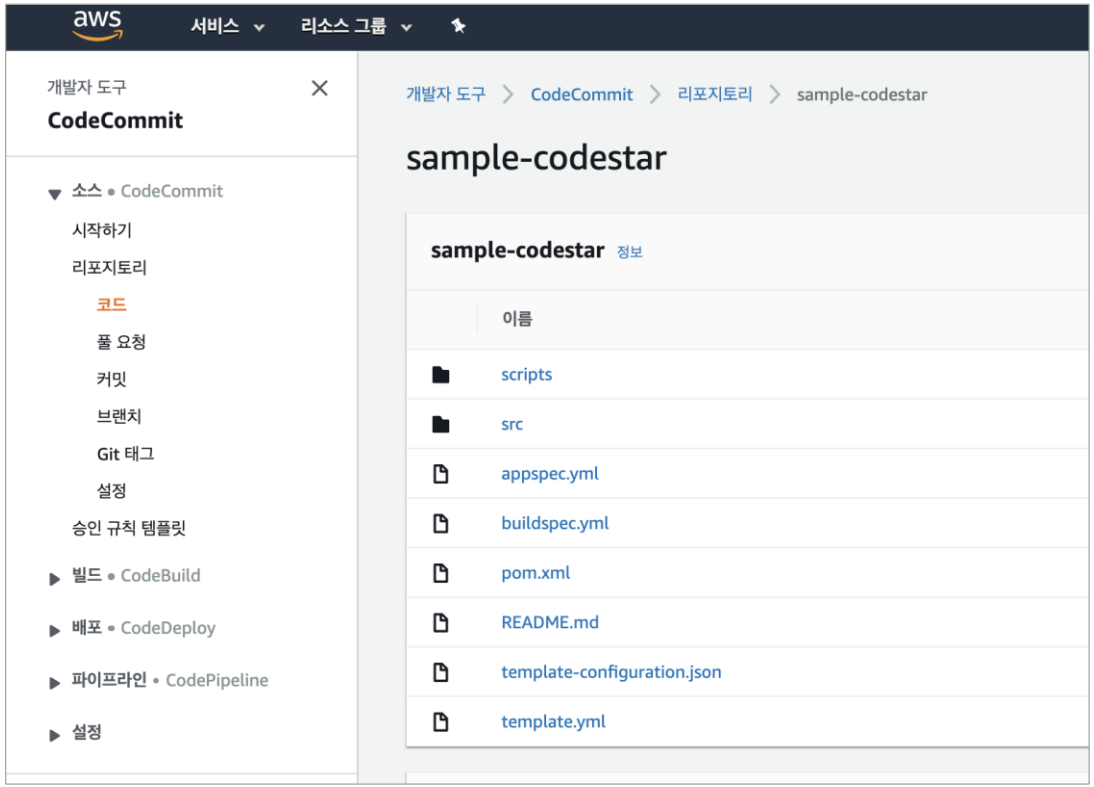



1. 자동으로 커밋-빌드-배포 로 이루어진 파이프라인이 생성됨
2. EC2 인스턴스를 생성하여 어플리케이션을 자동 배포함



# Aws DevOps – CodeCommit

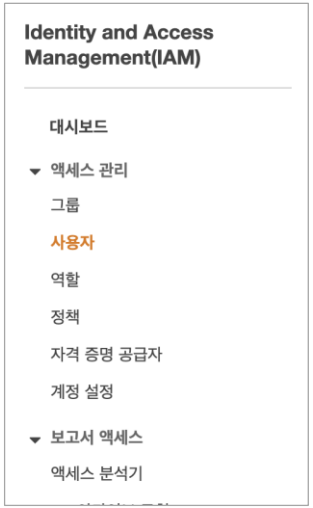
- AWS가 Managed Service로 제공하는 프라이빗 Git 리파지토리




1. Private Repository
2. Git 사용
3. 사용하기 위해서는 User 에 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





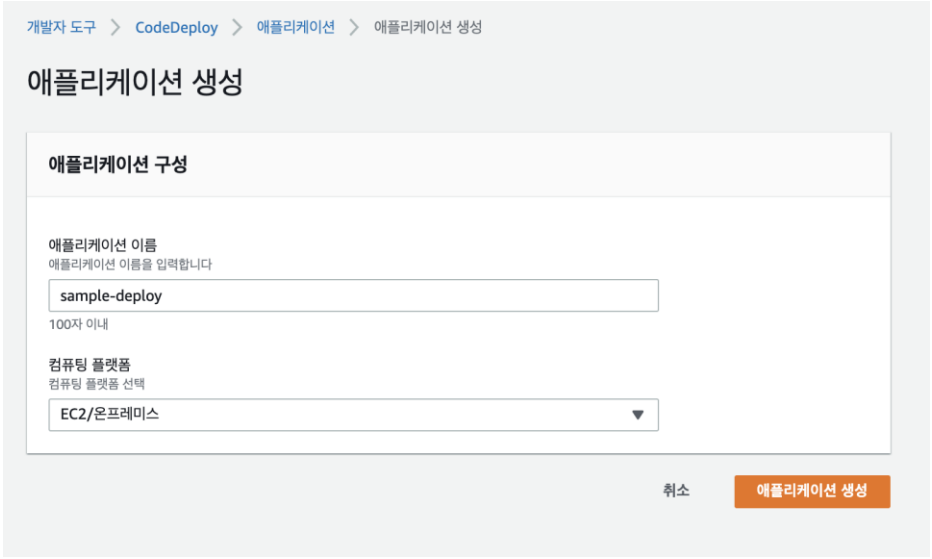
1. IAM 의 사용자 – 보안 자격 증명에서 CodeCommit 자격 생성

2. ID / PW 가 발급되면 해당 계정으로 CodeCommit의 git 접속 가능

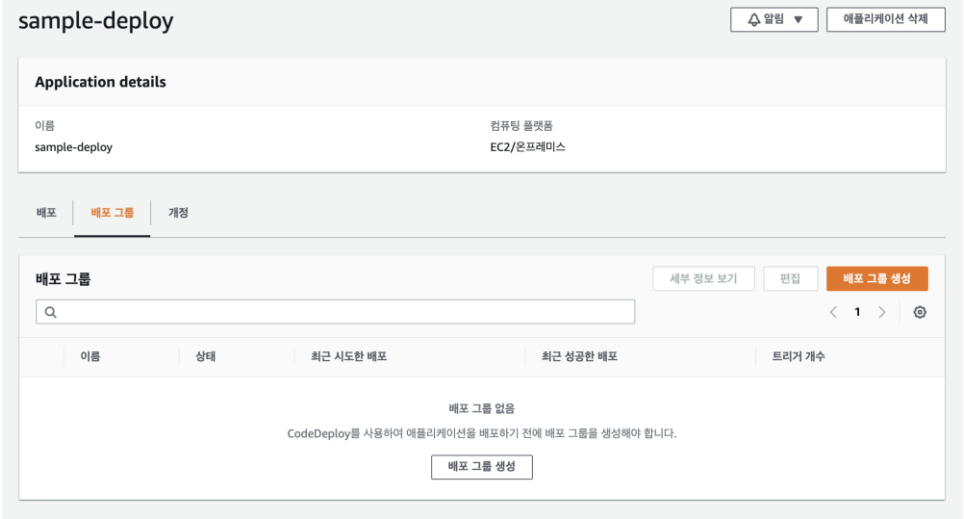
```
➜ MSA project git clone https://git-codecommit.ap-northeast-1.amazonaws.com/
Cloning into 'sample-codestar'...
Username for 'https://git-codecommit.ap-northeast-1.amazonaws.com': uengineDe
Password for 'https://uengineDev-at-979050235289@git-codecommit.ap-northeast-
remote: Counting objects: 44,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

# Aws DevOps – CodeDeploy

- AWS가 Managed Service로 제공하는 배포 자동화 도구 (05/20 현재, EKS 배포 미지원)



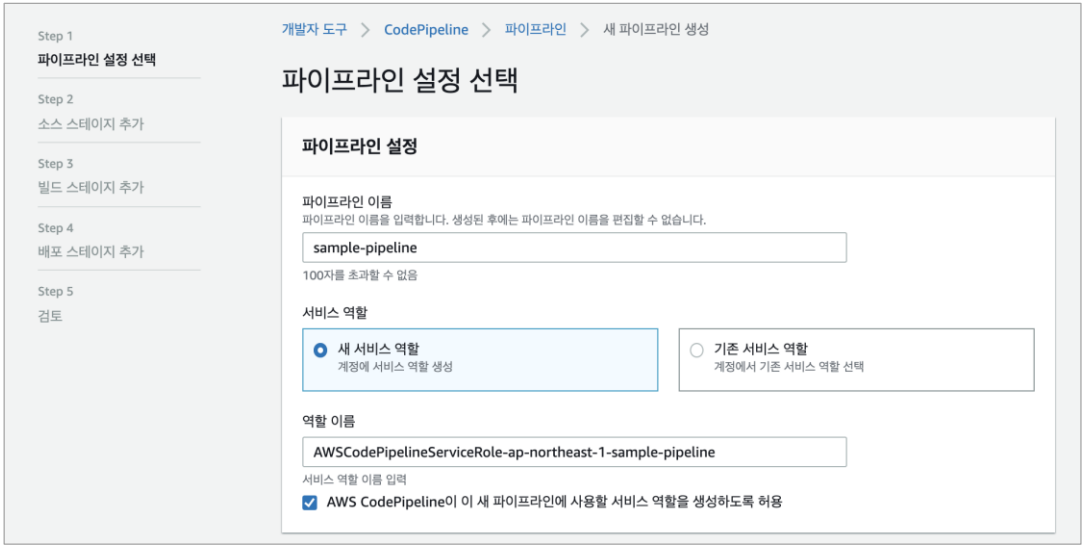

1. 배포 어플리케이션 – 배포그룹 생성 순으로 진행
2. 배포 그룹에서 블루/그린 디플로이가 가능함
3. 아직 EKS 를 지원 안함

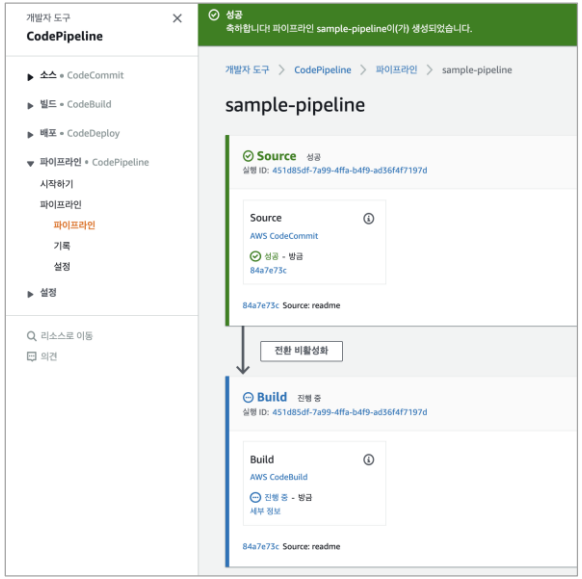


# Aws DevOps - CodePipeline

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

1. CodeStar 를 사용하면 자동으로 파이프라인 생성
2. 직접 만들기 위해서는 빌드/배포 단계를 생성해야 함



# Table of content

# 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.

### 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.

### 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.

### 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 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 기존 CI/CD 워크플로에 통합하는 것도 가능합니다. 예를 들어 기존 Jenkins 서버 설정에서 CodeBuild를 작업자 노드로 사용하여 빌드를 분산할 수 있습니다.

### 보안

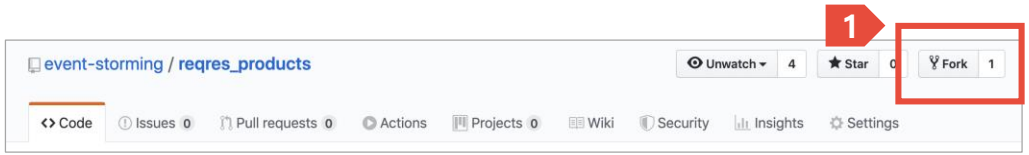
AWS CodeBuild에서는 고객이 지정하고 AWS Key Management Service(KMS)로 관리되는 키를 사용해 빌드 아티팩트가 암호화됩니다. CodeBuild는 AWS Identity and Access Management(IAM)에 통합되므로 사용자 지정 권한을 빌드 프로젝트에 할당할 수도 있습니다.

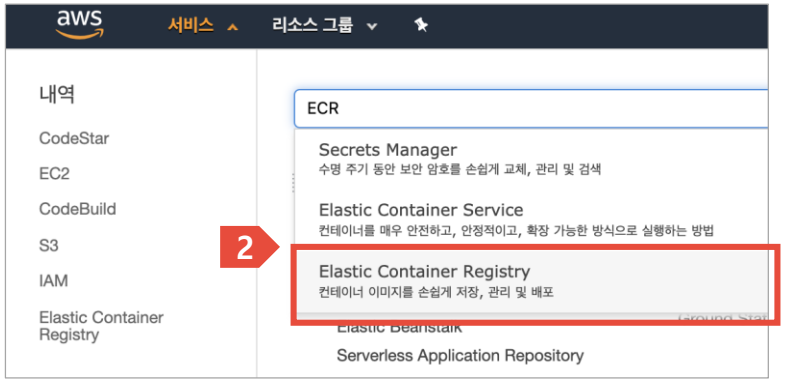
https://aws.amazon.com/ko/codebuild/

# 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

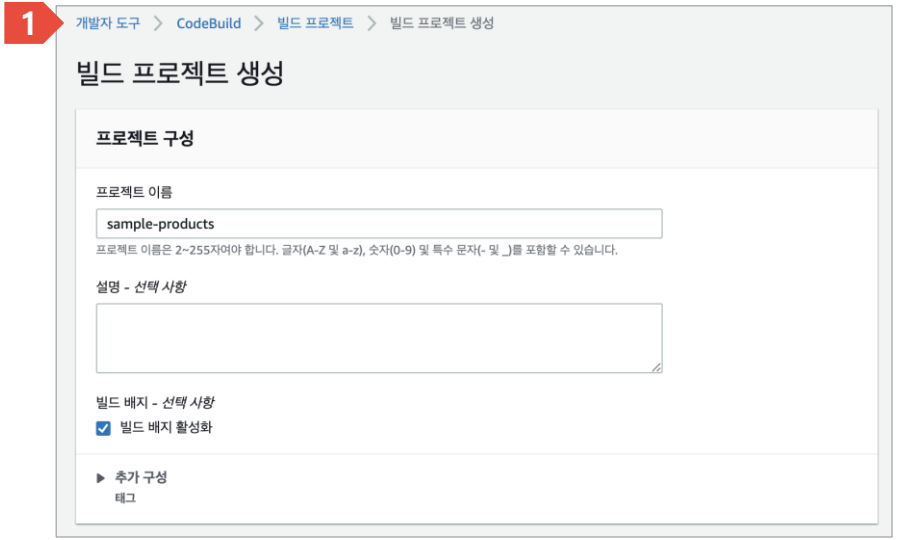
1. Github 에 있는 소스 fork https://github.com/event-storming/reqres_products
2. Docker image 를 저장하기 위한 공간인 ECR 에 “user00-products” 라는 Repository 생성성
3. Fork 받은 리파지토리의 Root에서 buildspec.yml을 편집하여 5행의 이미지명 수정

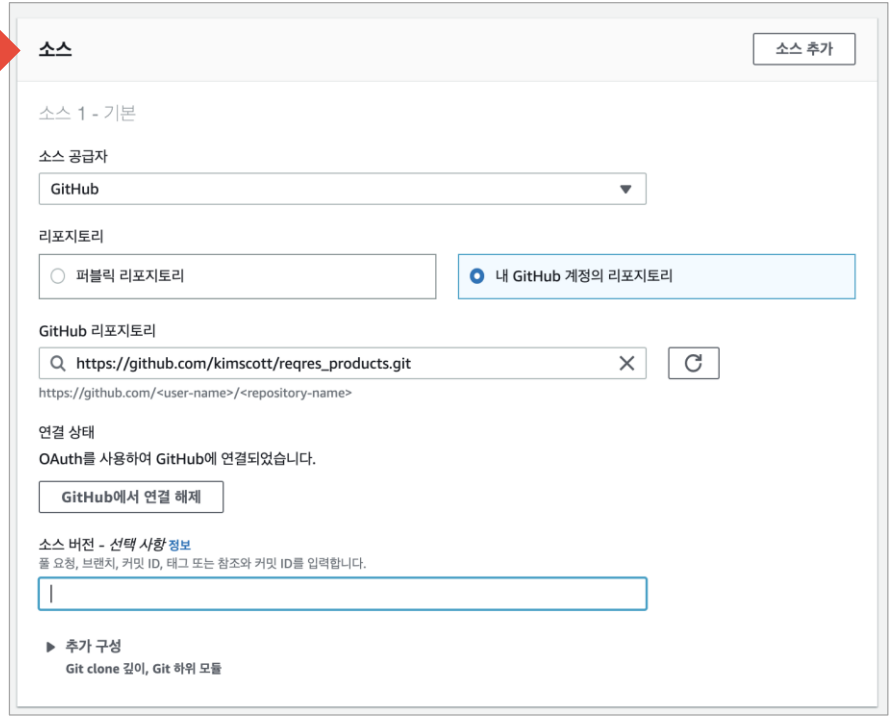


# CodeBuild : 생성 설정(1/3)

1. CodeBuild 프로젝트 생성
2. Github 계정 연결 후 Fork 한 리포지토리 연결



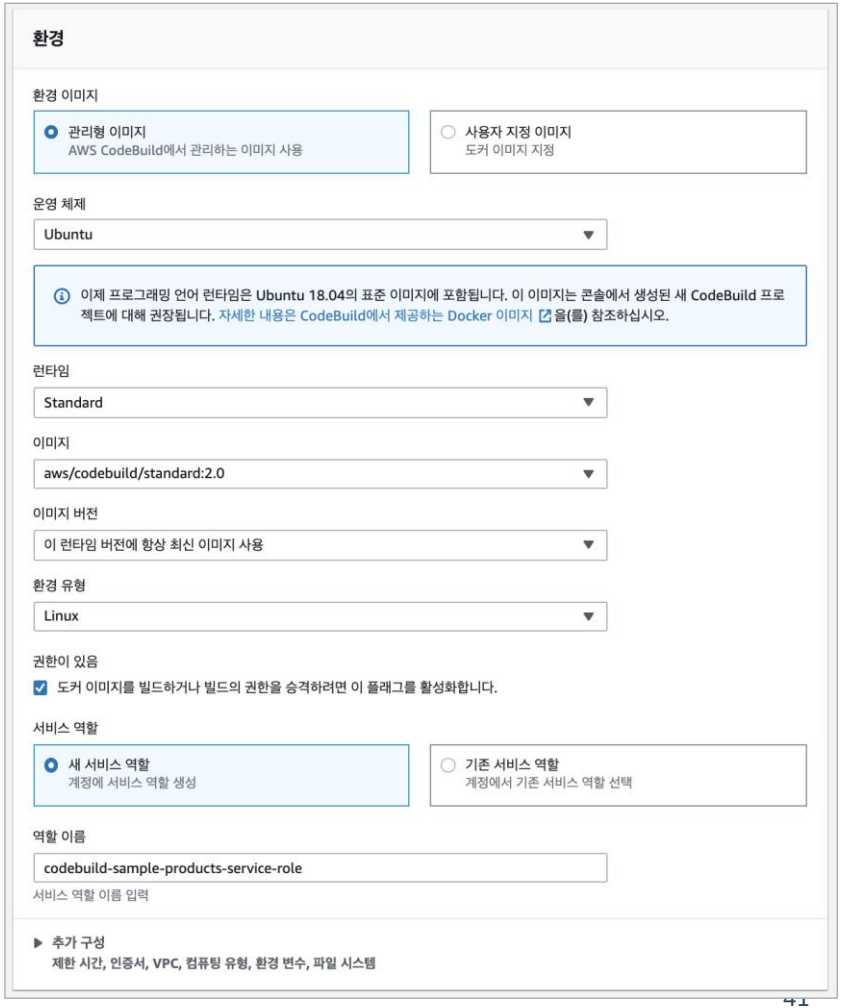



# CodeBuild : 생성 설정(2/3)

1. Webhook 을 설정하여 Github에 코드가 푸쉬될 때마다 트리거 동작
2. 빌드가 돌아갈 환경 설정
   - 운영체제 : 리눅스
   - 런타임 : Standard
   - 이미지 : Standard2.0 (이미지별로 환경이 다르니 주의)
   - 환경유형 : Linux
   - 도커 권한 체크 필수

1


2

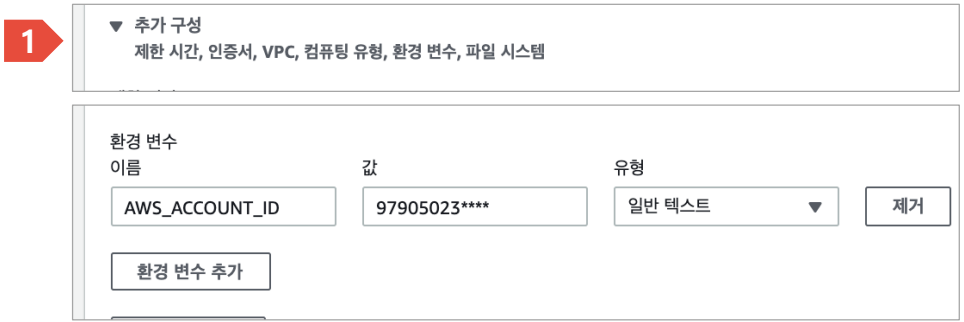


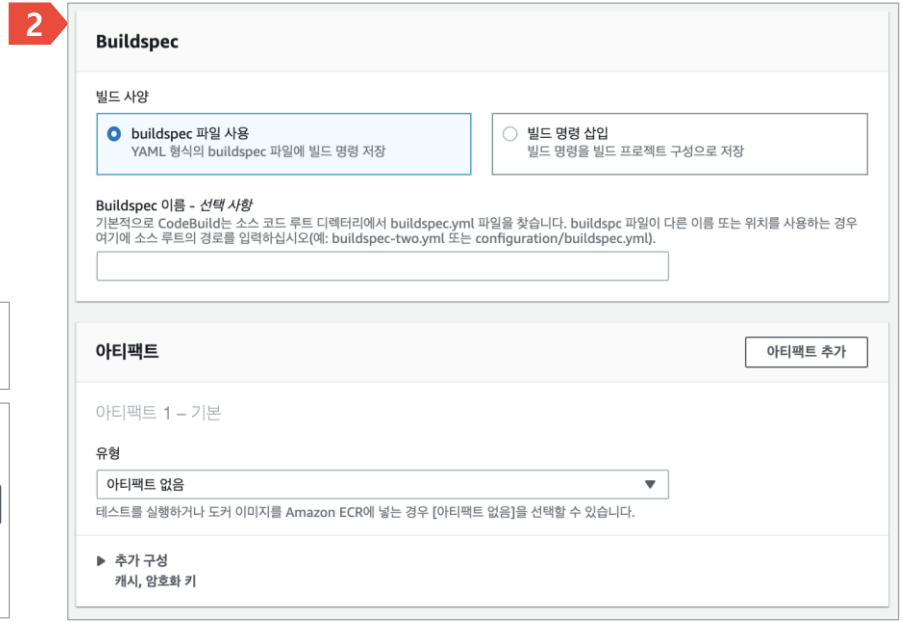

✓ 런타임 이미지 참조문서 : https://docs.aws.amazon.com/ko_kr/codebuild/latest/userguide/build-env-ref-available.html

# CodeBuild : 생성 설정(3/3)

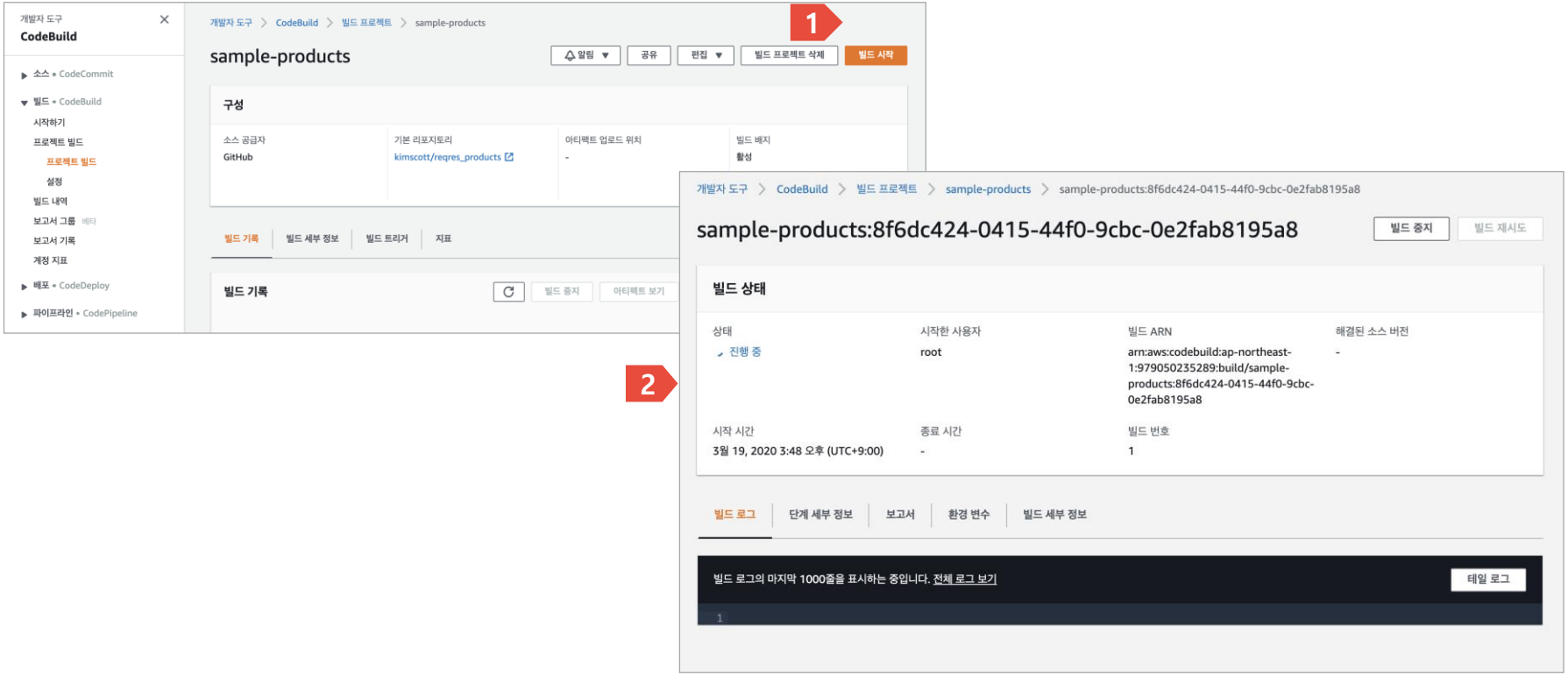
1. 추가 구성에 환경 변수값을 입력 : AWS 계정 ID
   - AWS_ACCOUNT_ID
   - ECR 에서 이미지 주소의 가장 앞에 값임
   - echo $(aws sts get-caller-identity --query Account --output text) 명령으로 조회 가능

2. buildspec.yml 파일을 사용 하겠다고 체크 후, 저장



# CodeBuild : 빌드 시작



# CodeBuild : buildspec.yml

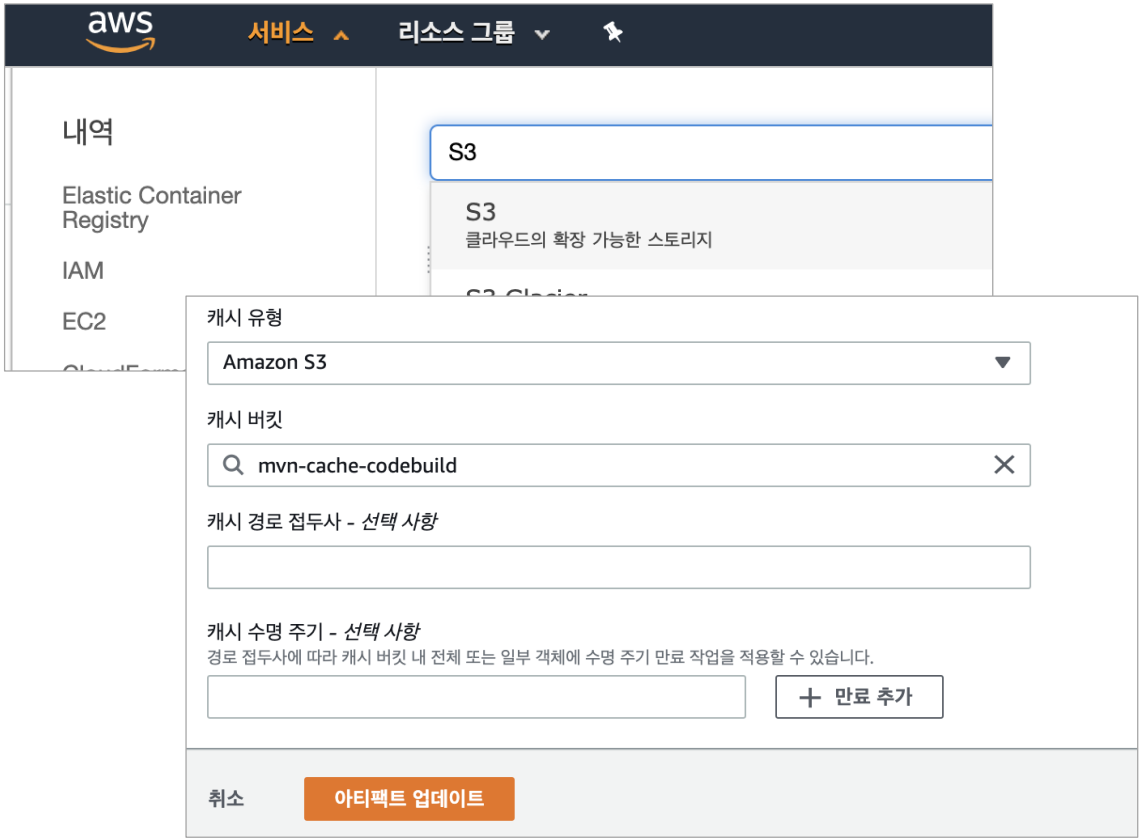
```yaml
version: 0.2

env:
  variables:
    IMAGE_REPO_NAME: "products"

phases:
  install:
    runtime-versions:
      java: openjdk8
#       docker: 18
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - echo $IMAGE_REPO_NAME
      - echo $AWS_ACCOUNT_ID
      - echo $AWS_DEFAULT_REGION
      - echo $CODEBUILD_RESOLVED_SOURCE_VERSION
      - echo start command
#       - $(aws ecr get-login --no-include-email --region $AWS_DEFAULT_REGION)
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - mvn package -Dmaven.test.skip=true
#       - docker build -t $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION .
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker image...
#       - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$IMAGE_REPO_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION

# cache:
#   paths:
#     - '/root/.m2/**/*'
```

1. buildspec 에 대한 상세 가이드 - https://docs.aws.amazon.com/ko_kr/codebuild/latest/userguide/build-spec-ref.html

2. phases: 필수 시퀀스입니다. 빌드의 각 단계 동안 CodeBuild가 실행하는 명령을 나타냅니다.

3. Install, pre_build, build, post_build 4개의 시퀀스를 적절히 사용할 수 있음

# CodeBuild : cache 적용



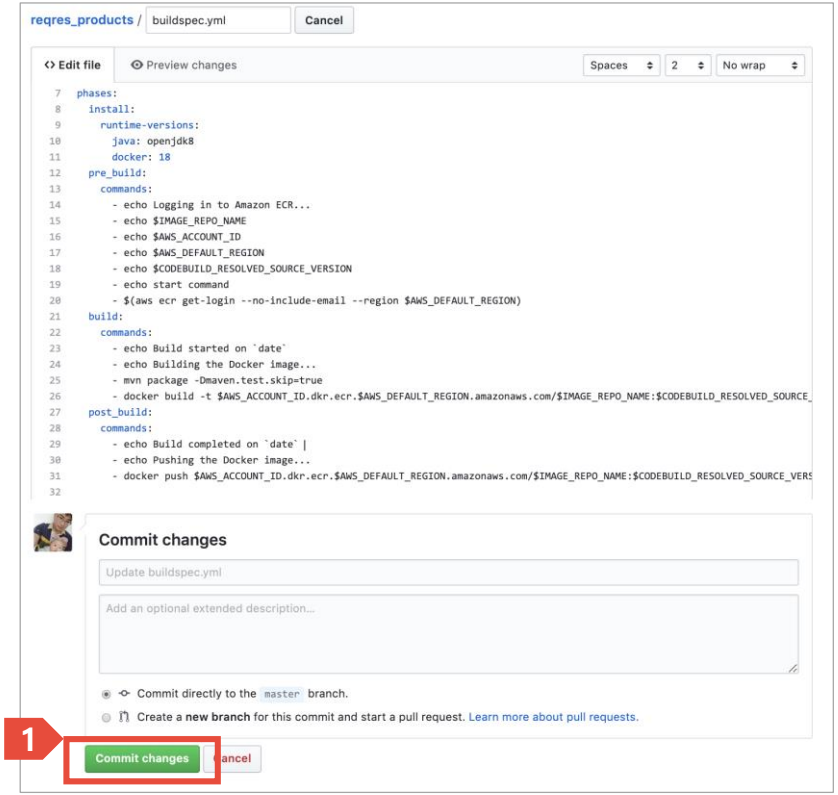

1. Maven 으로 빌드시, 라이브러리를 캐쉬화 시킴 (캐쉬가 없으면 빌드할때마다 약 5분정도 소요됨)
2. 캐쉬를 저장할 S3 스토리지 생성
3. 버킷 생성 – 리전을 CodeBuild 생성 리전과 일치 해야함
4. 빌드 선택 – 아티팩트 편집
5. 추가구성 – 캐시 유형을 S3 로 선택 후 버킷 선택
6. buildspec.yml 을 에서 cache 부분 주석 해제 후 커밋

# CodeBuild : ECR 연결



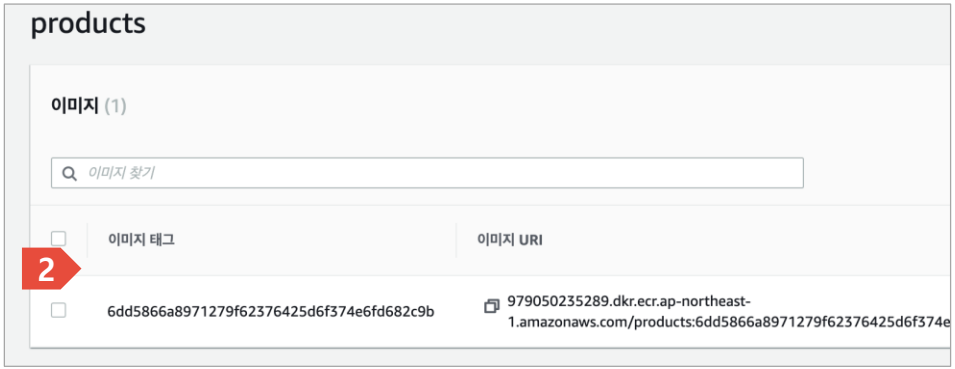
1. 프로젝트 빌드 - 빌드 선택 - 빌드 세부정보 - 환경 - 서비스 역할 클릭하여 IAM 역할 페이지로 이동
2. 인라인 정책 추가 > json 하여 ECR 관련 정책을 추가
3. 관련 스크립트는 WorkFlowy 에 “CodeBuild 와 ECR 연결 정책” 검색
4. 정책명 입력 후, user00-codebuild-policy 검토 및 생성

# CodeBuild : Docker Build/Push




1. buildspec.yml 파일의 Docker 관련 주석을 모두 해제 후, 코드 커밋/푸쉬

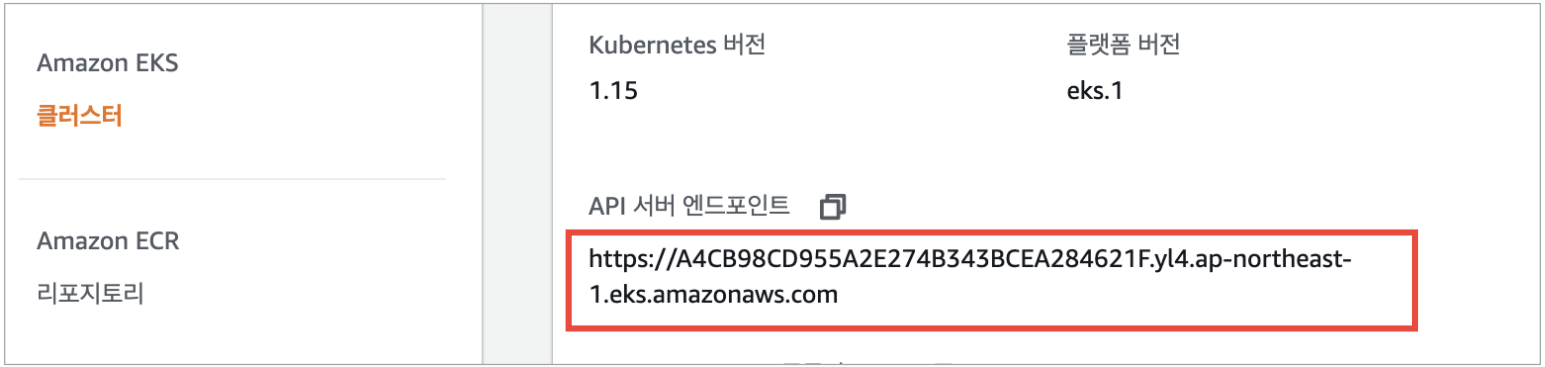
2. 빌드 성공 후 ECR 에 Docker Image 가 정상적으로 생성 되었는지 확인



# CodeBuild : EKS 연결 (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



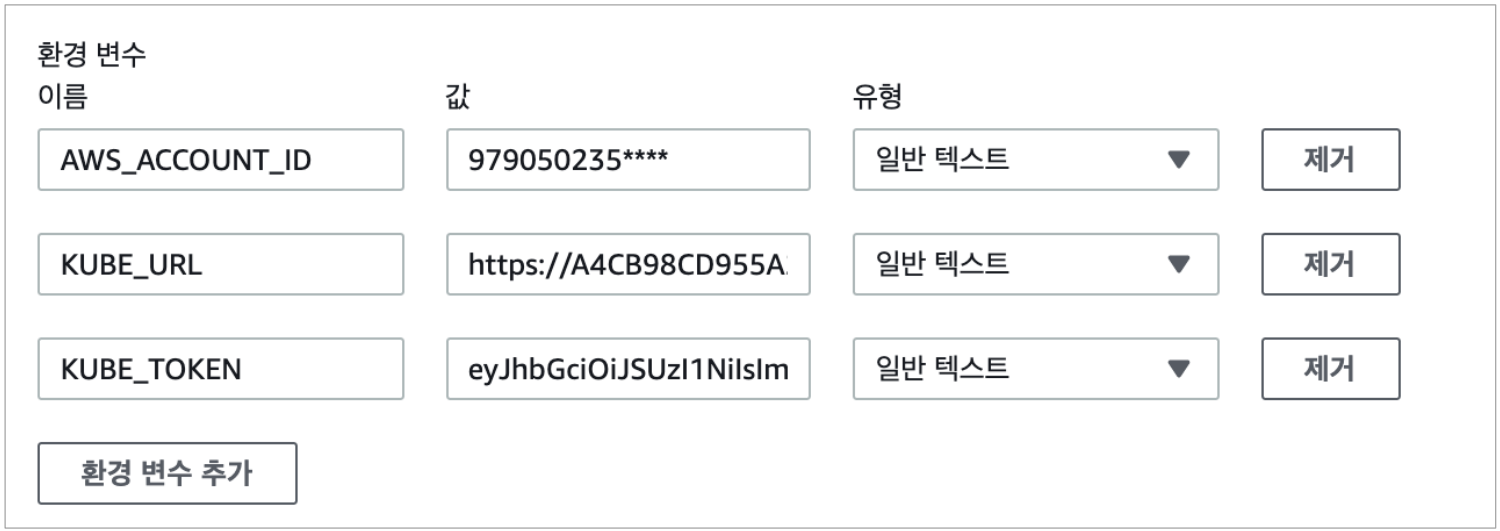
# CodeBuild : EKS 연결 (2/4)

1. 쿠버네티스 토큰 생성은 ServiceAccount 생성 후 해당 SA 에 cluster-admin 롤을 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 | a
wk '{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.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJ
uZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY29
```

# CodeBuild : EKS 연결 (3/4)

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



# CodeBuild : EKS 연결 (4/4)

1. Github 소스의 root 에 buildspec-kubectl.yml 파일의 내용을 복사하여 buildspec.yml 파일에 붙여 넣은 후 커밋/푸쉬 한다.

2. CodeBuild 가 자동 실행 되어, EKS 클러스 배포 후 서비스가 정상적으로 기동 되었는지 확인 한다.

1

```
post_build:
  commands:
    - echo Pushing the Docker image...
    - docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_DEFAULT_REGION.amazonaws.com/$_PROJECT_NAME:$CODEBUILD_RESOLVED_SOURCE_VERSION
    - echo connect kubectl
    - kubectl config set-cluster k8s --server="$KUBE_URL" --insecure-skip-tls-verify=true
    - kubectl config set-credentials admin --token="$KUBE_TOKEN"
    - kubectl config set-context default --cluster=k8s --user=admin
    - kubectl config use-context default
    - |
      cat <<EOF | kubectl apply -f -
      apiVersion: v1
      kind: Service
      metadata:
```

2

```
➜  ~ kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
orders-66f47b6455-pp6dh     1/1     Running   0          40h
products-dbc86b85b-skblg    1/1     Running   0          2m41s
```

# AWS CodeBuild를 활용한 CI/CD 2nd Lab.

**From Page.38**

1. Github 에 있는 소스 fork
   https://github.com/event-storming/reqres_delivery

2. Docker image를 저장하기 위한 공간인 ECR 에 “user00-delivery” 라는 Repository 생성

3. reqres_products 레파지토리에서 buildspec.yml 파일 내용을 복사해 오기

4. Fork 받은 리파지토리의 Root에서  buildspec.yml을 편집하여 5행의 이미지명 수정
   - user00-delivery

# CodeBuild : 활용 – set Webhook IGNORE (1/2)

1. 환경 별로 다른 빌드를 하고 싶으면 빌드 파일을 다르게 생성하여 빌드 프로젝트를 각자 생성 하면 된다.
   (예 : 개발 환경 – buildspec-dev.yml, 운영환경 - buildspec-prod.yml)

2. 기존 프로젝트의 webhook 트리거를 잠시 제거하고, buildspec-sample.yml 파일로 빌드 프로젝트를 생성 한다.

3. 특정 파일 수정일때 트리거 작동 안하도록 설정

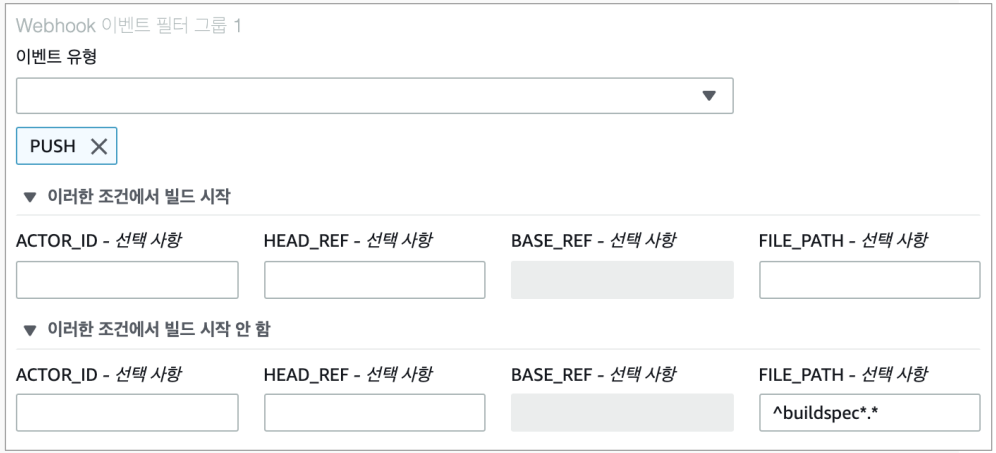
   - Webhook 이벤트 항목을 편집한다.

   - 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)

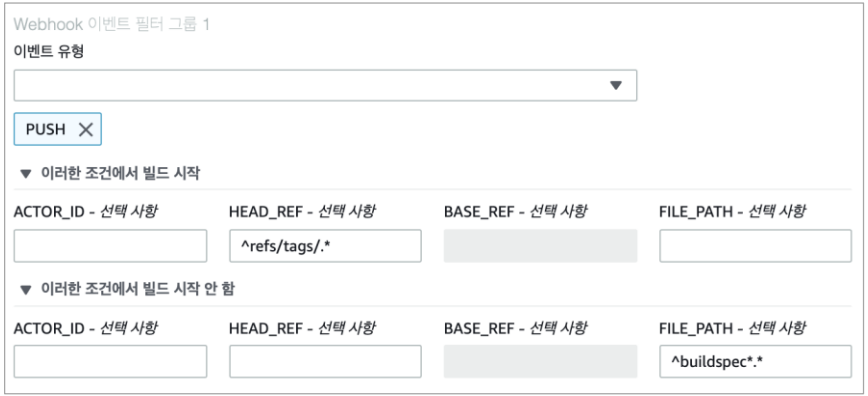


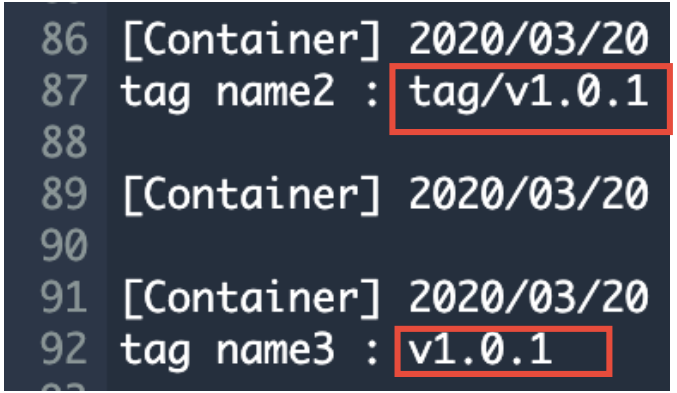
# CodeBuild : 활용 – set Webhook on GIT Tags (2/2)

1. 태그 이벤트일때 트리거 작동하도록 설정

   - 이러한 조건에서 빌드 시작 항목 – HEAD_REF 에 “^refs/tags/.*” 를 입력한다.
     > 팁 : (아래 이미지 처럼 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 브렌치에서만 작동하는것을 연습해보자.

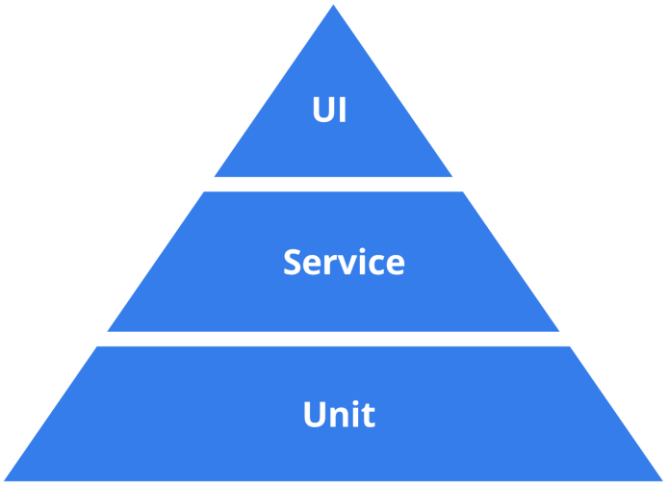


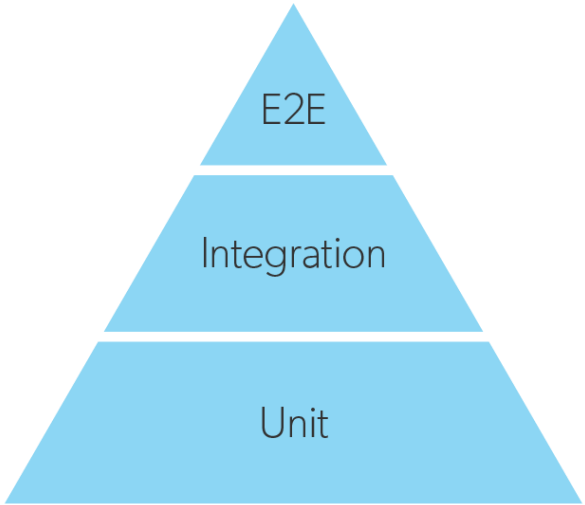
# Table of content

# Test 전략

항상 성공 할수 있는것, 항상 동일한 결과를 나오는 것을 테스트 해야함




기준 – 테스트 대상




기준 – 테스트 범위



56

# MSA Test 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

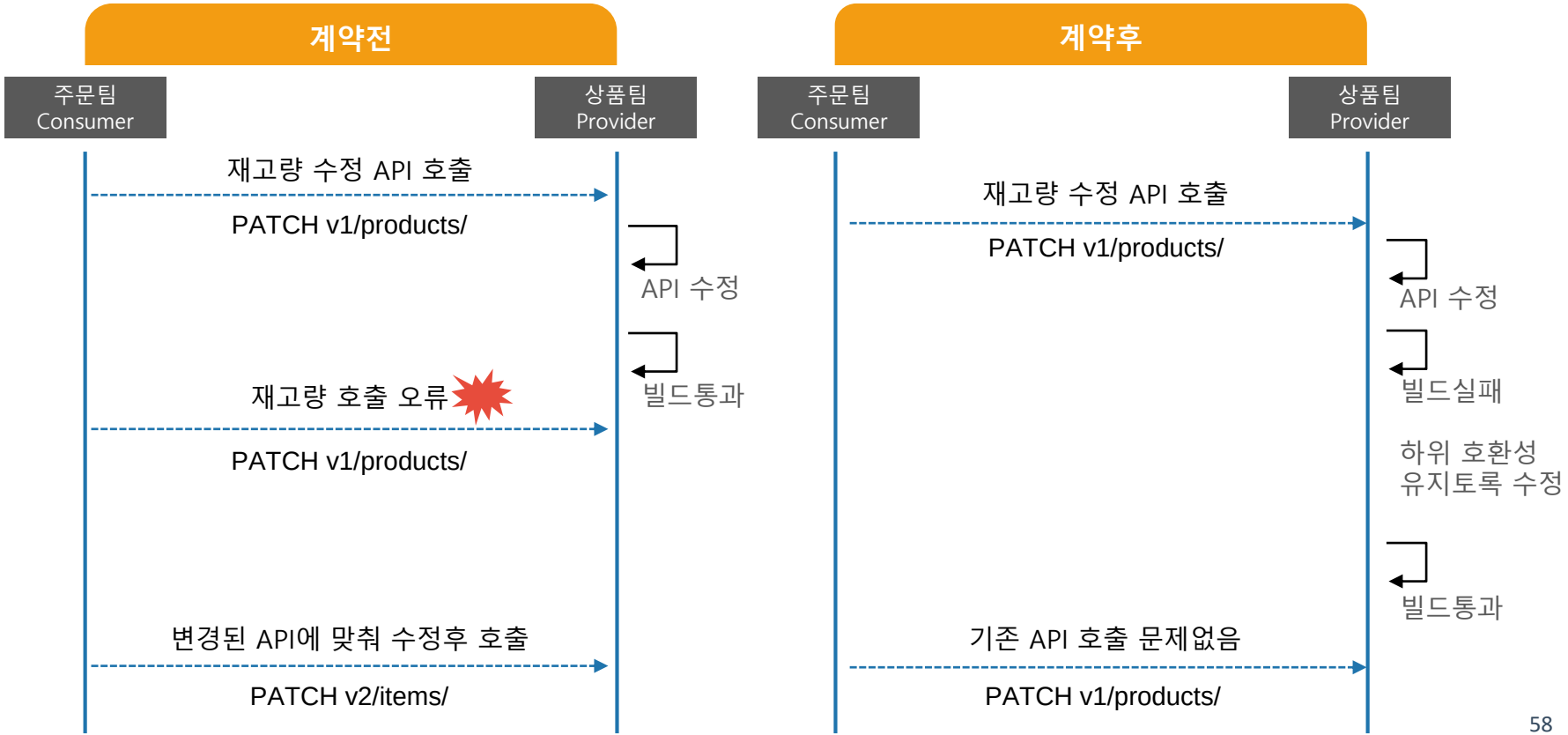
   > **API Test** : 개발된 API 가 정상 작동하는지 테스트. 테스트를 위한 서비스를 생성 후, 테스트 후, 서비스 제거 방식으로 실행

   > **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

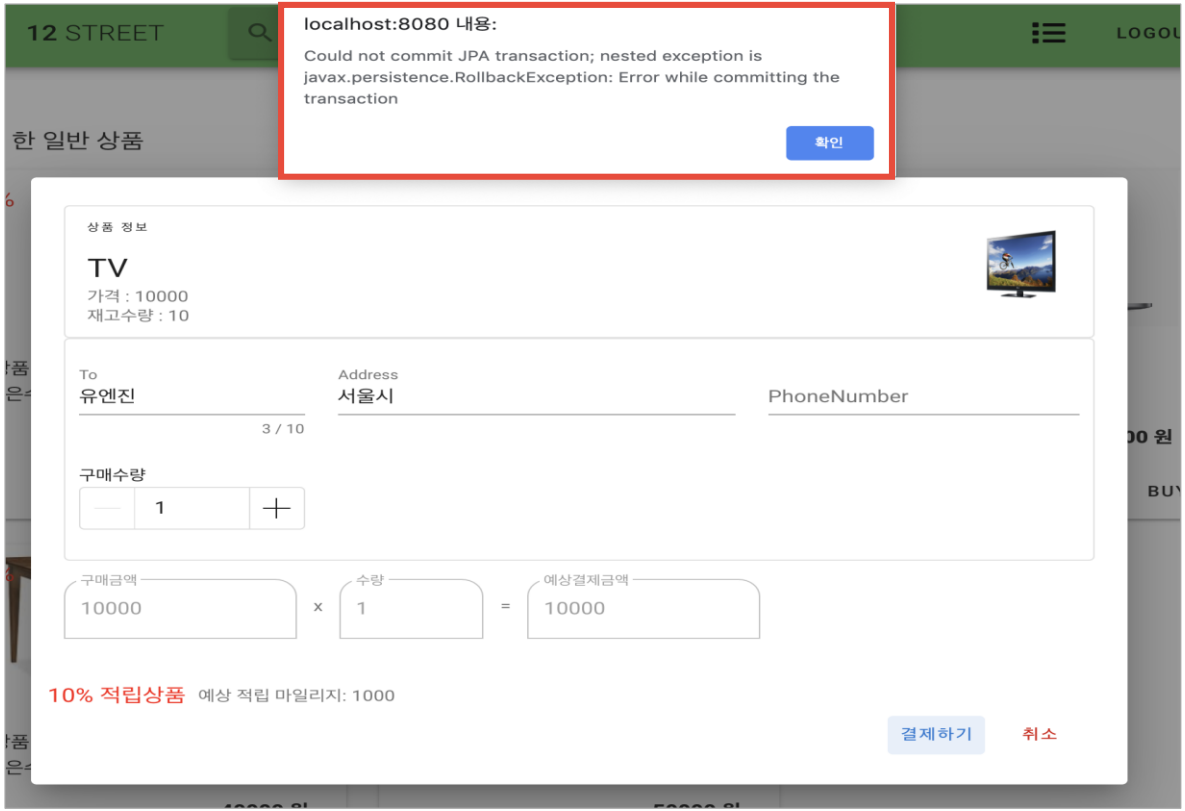
소스 위치

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

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

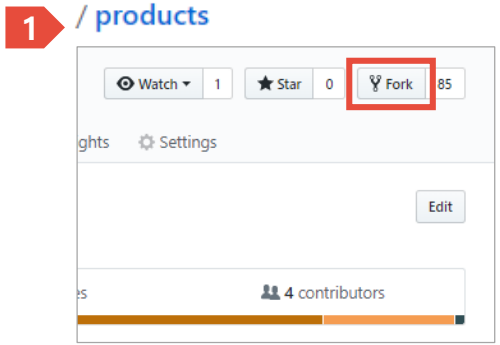
/item/{productId} 로 변경

# 주문팀의 서비스 장애 발생

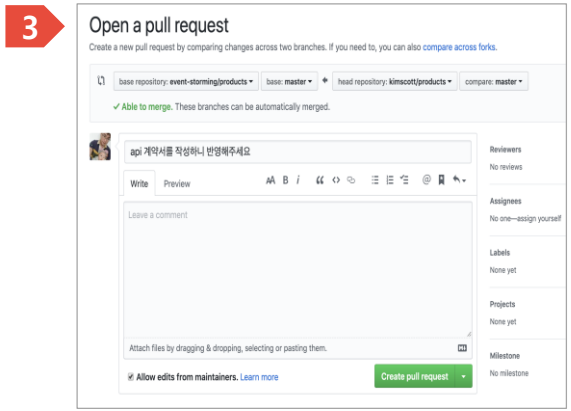


# 계약체결(1/2) - 주문팀에서 계약서 작성 후, 상품팀에 체결 요청

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

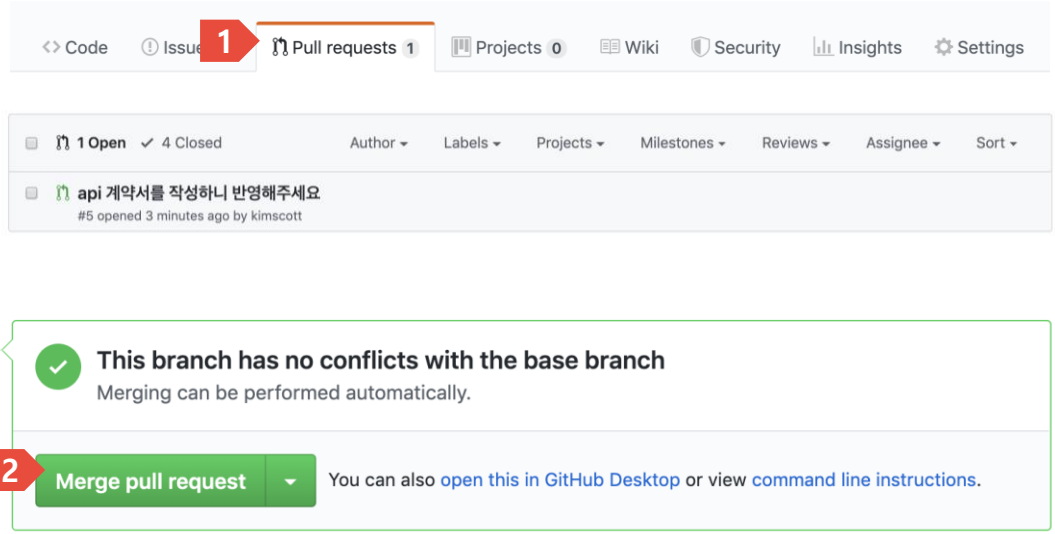






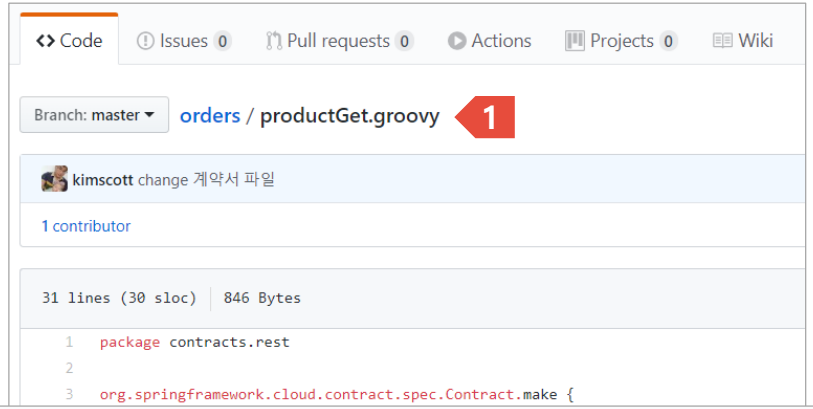
# 계약체결(2/2) - 상품팀에서 계약서 수락

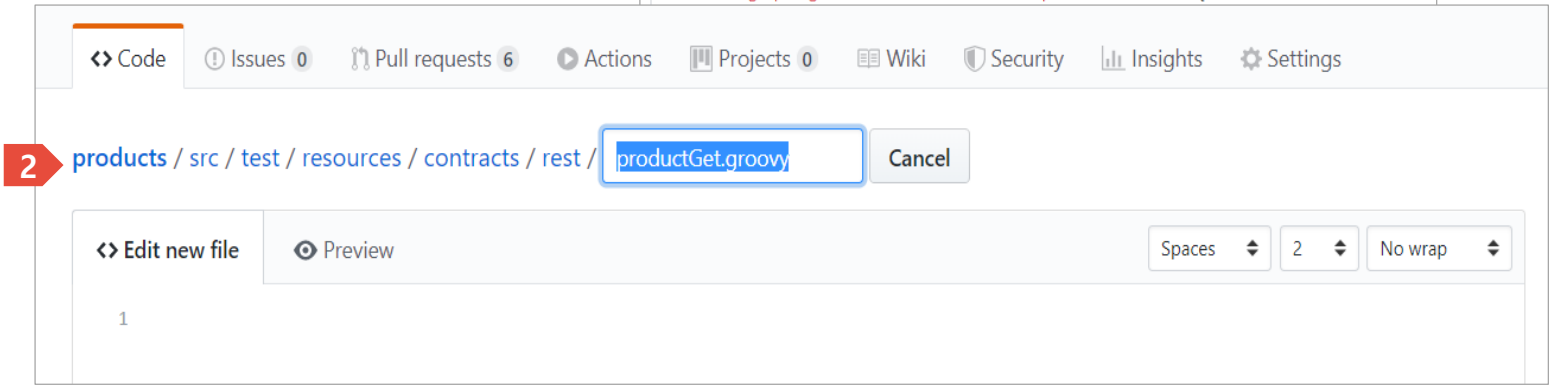
- 상품팀 : Pull Request 메뉴 선택
- 상품팀은 해당 계약서를 accept 하여 반영함
- 상품 서비스는 이제부터는 계약서를 안지켰을때 아예 배포가 안됨



# 계약체결 WORKAROUND

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



# 계약체결 후, 상품팀은 계약 위반으로 배포 실패함

mvn package 단계 실패

실패함 시작 시간: 5분 전 현재 단계: COMPLETED

```
7858 Hibernate:
7859
7860     drop sequence if exists hibernate_sequence
7861 2020-03-27 07:14:32.398 TRACE 563 --- [       Thread-5] o.h.type.spi.TypeConfiguration$Scope     : Handling #sessionFactoryClosed from
     [org.hibernate.internal.SessionFactoryImpl@45eab322] for TypeConfiguration
7862 2020-03-27 07:14:32.400 DEBUG 563 --- [       Thread-5] o.h.type.spi.TypeConfiguration$Scope     : Un-scoping TypeConfiguration
     [org.hibernate.type.spi.TypeConfiguration$Scope@2ba066c] from SessionFactory [org.hibernate.internal.SessionFactoryImpl@45eab322]
7863 2020-03-27 07:14:32.402  INFO 563 --- [       Thread-5] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown initiated...
7864 2020-03-27 07:14:32.409  INFO 563 --- [       Thread-5] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Shutdown completed.
7865 [INFO]
7866 [INFO] Results:
7867 [INFO]
7868 [ERROR] Failures:
7869 [ERROR]   RestTest.validate_productGet:33
7870 Expecting:
7871  <404>
7872 to be equal to:
7873  <200>
7874 but was not.
7875 [INFO]
7876 [ERROR] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0
7877 [INFO]
7878 [INFO] ------------------------------------------------------------------------
7879 [INFO] BUILD FAILURE
7880 [INFO] ------------------------------------------------------------------------
7881 [INFO] Total time:  03:56 min
7882 [INFO] Finished at: 2020-03-27T07:14:32Z
7883 [INFO] ------------------------------------------------------------------------
7884 [ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.22.1:test (default-test) on project boot-camp-products:
     There are test failures.
7885 [ERROR]
```

# 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

# Table of content

# Course Test

- 필기 테스트 문항으로 배포