CI/CD 이해

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 **자동화**와 **모니터링**을 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 통합할때의 문제점을 해결하고 자동화 시켜 지속적으로 유지 시키는 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 까지의 배포를 하여, 실제 사용자가 사용할수 있겠끔 하는 단계**

AUTOMATIC
MANUAL

- 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

Continuous delivery

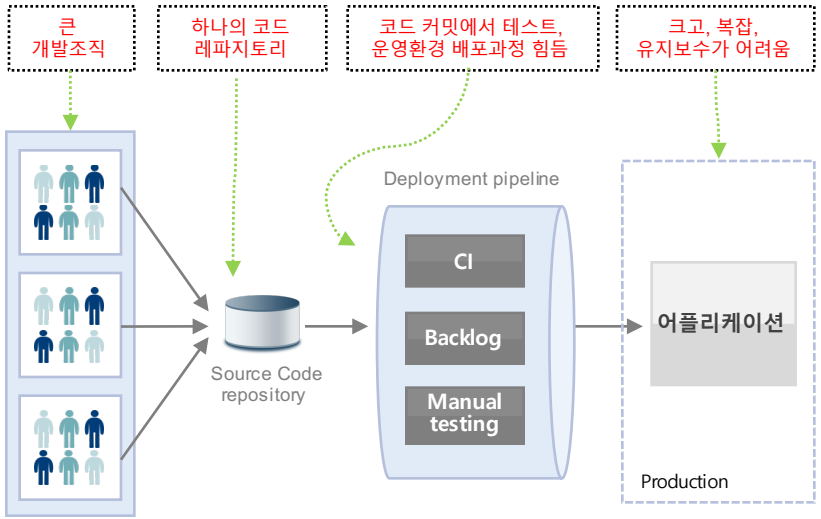
Continuous deployment

Table of content

Process Change : 열차말고 택시를 타라!

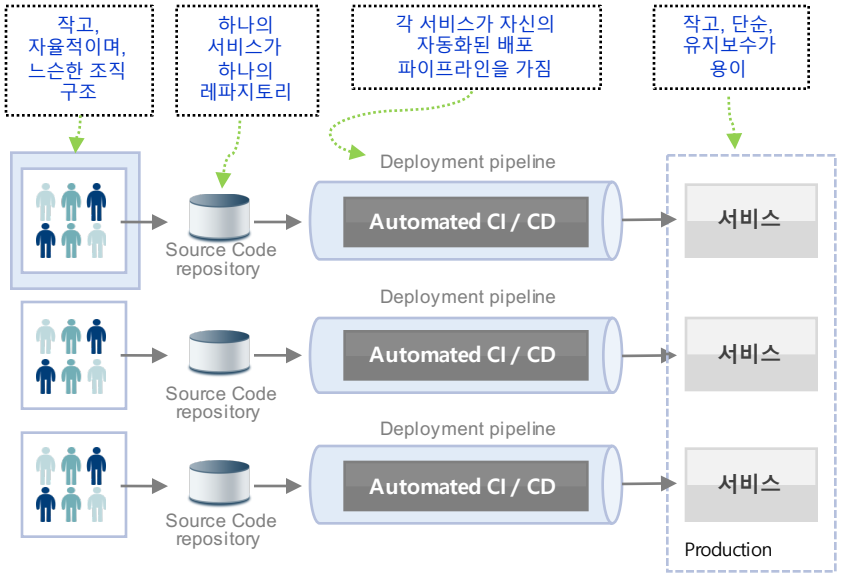
모노리식 개발 및 운영환경

- 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

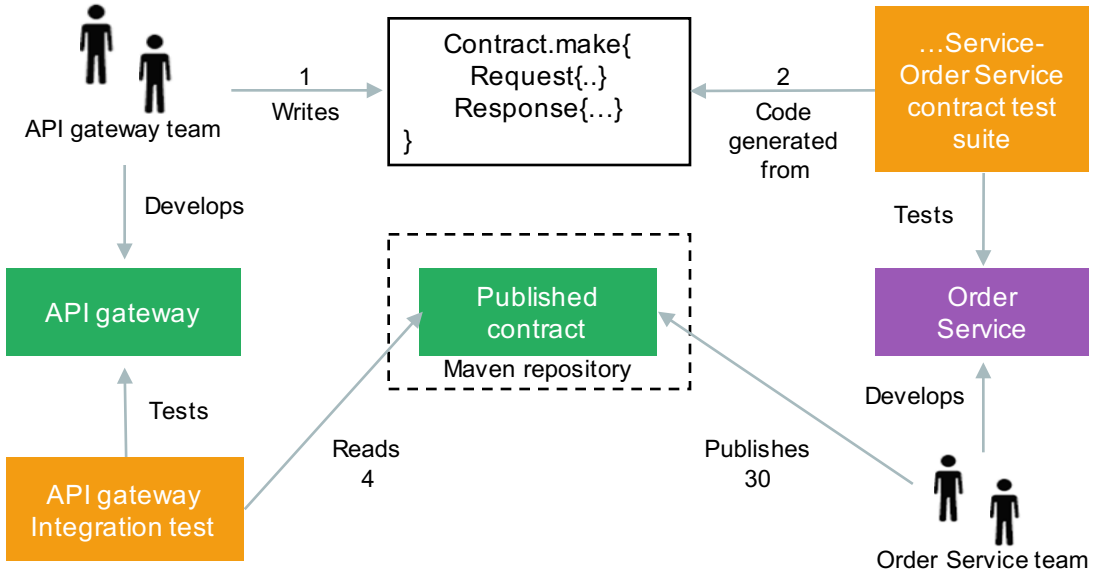


Process Change : Consumer-driven Test

MSA 서비스의 테스트

컨트랙트 테스트는 서비스 수요자가 주도하여 테스트를 작성한다.

수요자가 테스트를 작성하여 제공자의 레포지토리에 Pull-Request하면, 제공자가 이를 Accept 한후에 제공자측에서 테스트가 생성되어 테스트가 벌어진다.



특징

- MSA 테스트에는 Contract-based Test가 특징적
- API Consumer가 먼저 테스트 작성
- API Provider가 Pull Request 수신 후 테스트는 자동으로 생성됨
- Consumer 측에 Mock 객체가 자동생성 병렬 개발이 가능해짐
- Spring Cloud Contract가 이를 제공
- Provider의 일방적인 버전업에 따른 하위 호환성 위배의 원천적 방지

Continuous Delivery

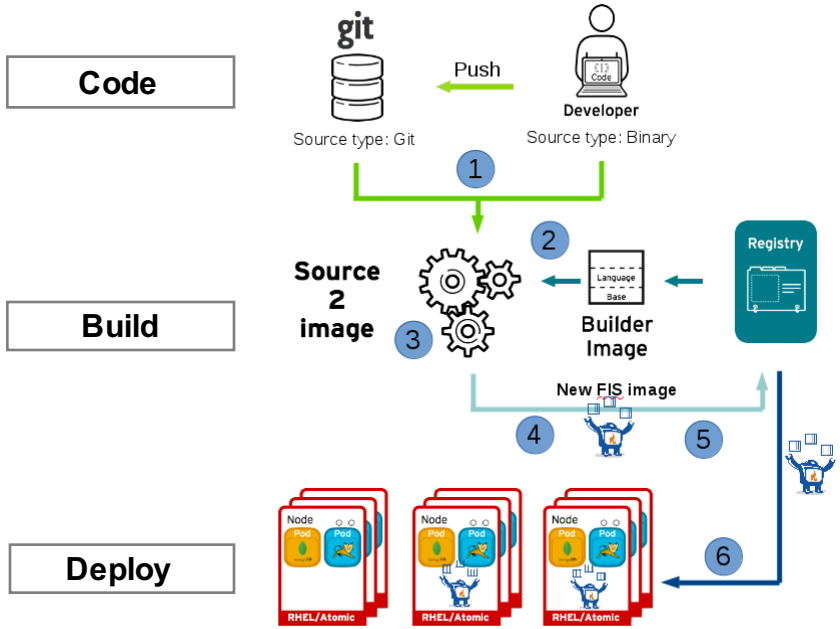
Amazon, Google, Netflix, Facebook, Twitter는 얼마나 자주 배포할까요?

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.

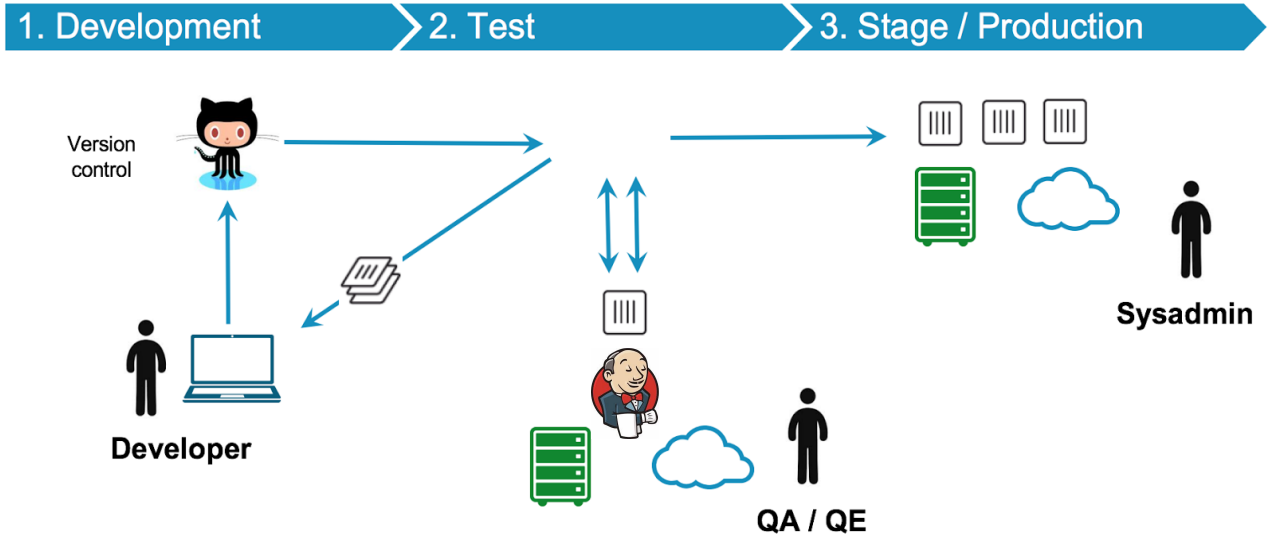
If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.

Container Solutions

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	✓	✓	✗	■■■	□□□	□□□	■■■

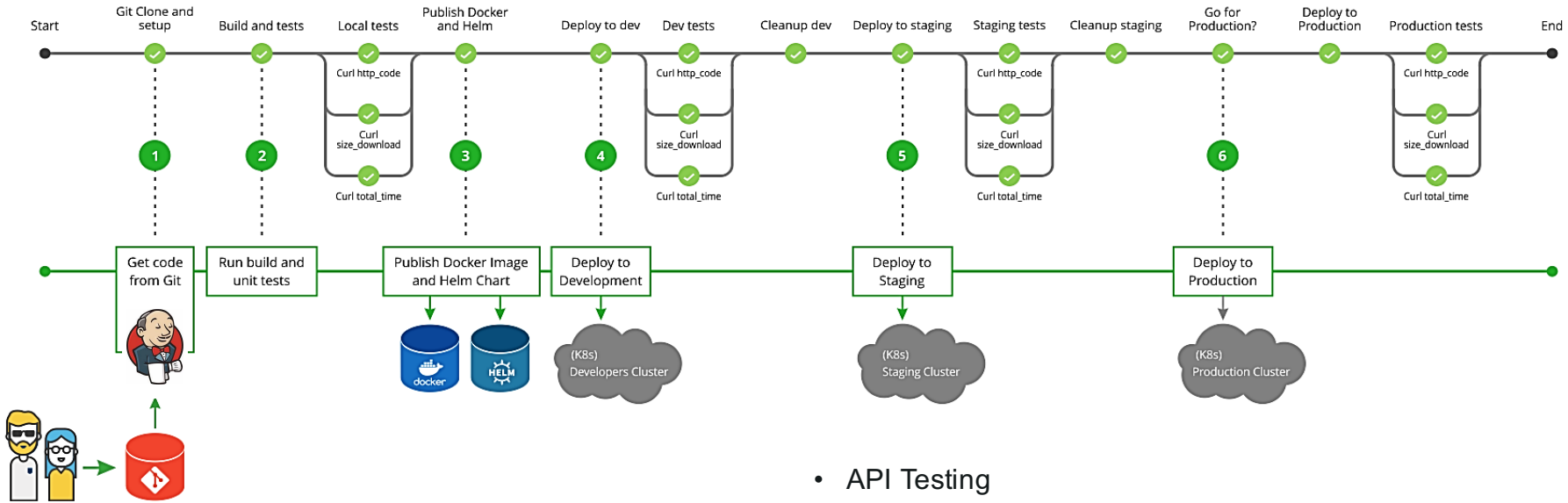
CI/CD Tools – conventional CI/CD

CI /CD Workflow



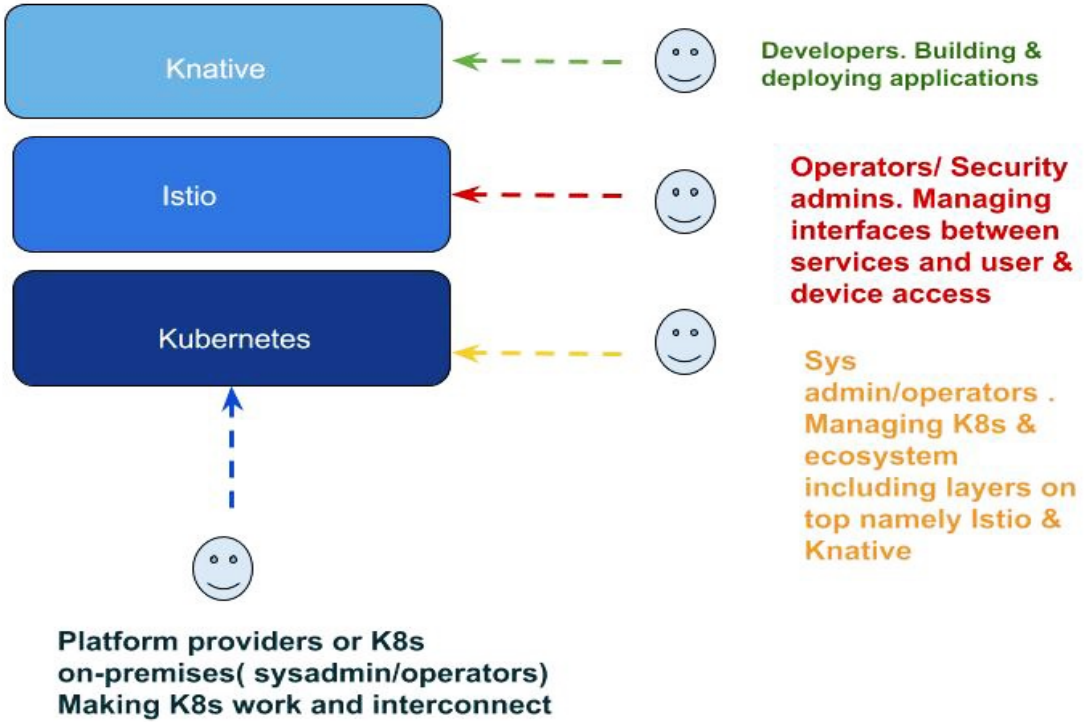
- 잦은 배포
- 일반 서버에 배포
- 테스트 자동화/커버리지
- 도구: Jenkins, Travis, Gitlab 등

CI/CD Tools – Container-based



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

CI/CD Tools - Serverless



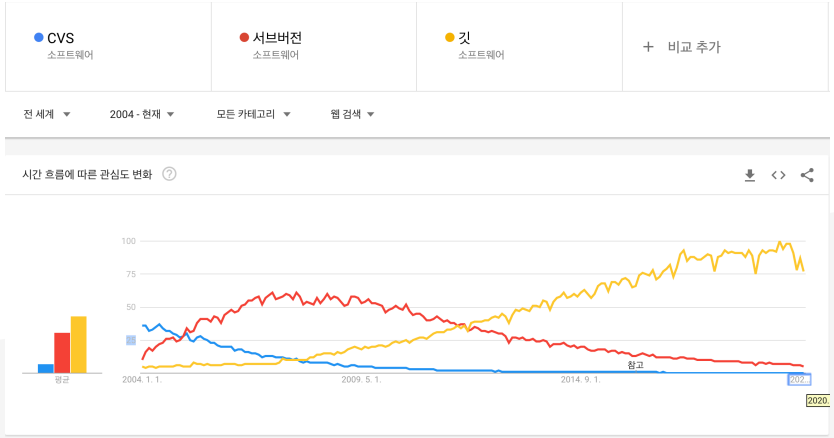
- Infrastructure As A Code
- Application Configuration 과 Infra Configuration을 하나의 설정에 통합
- 단일 도구

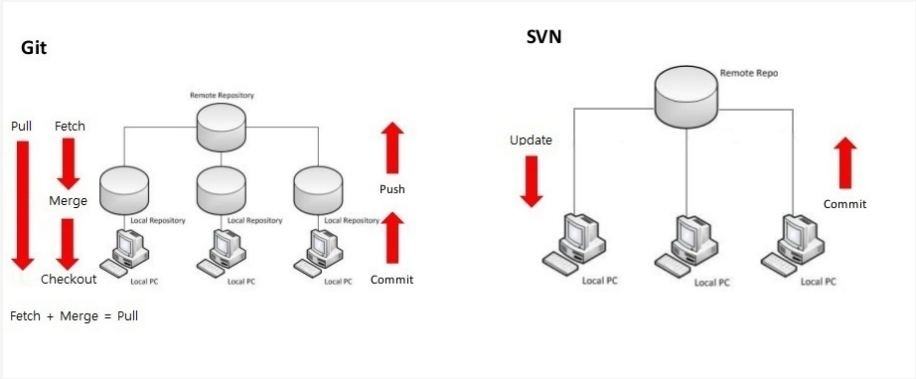
Table of content

형상관리 - History

- 형상관리란? :
  소스의 변화를 끊임없이 추적하고, 버전별로 관리

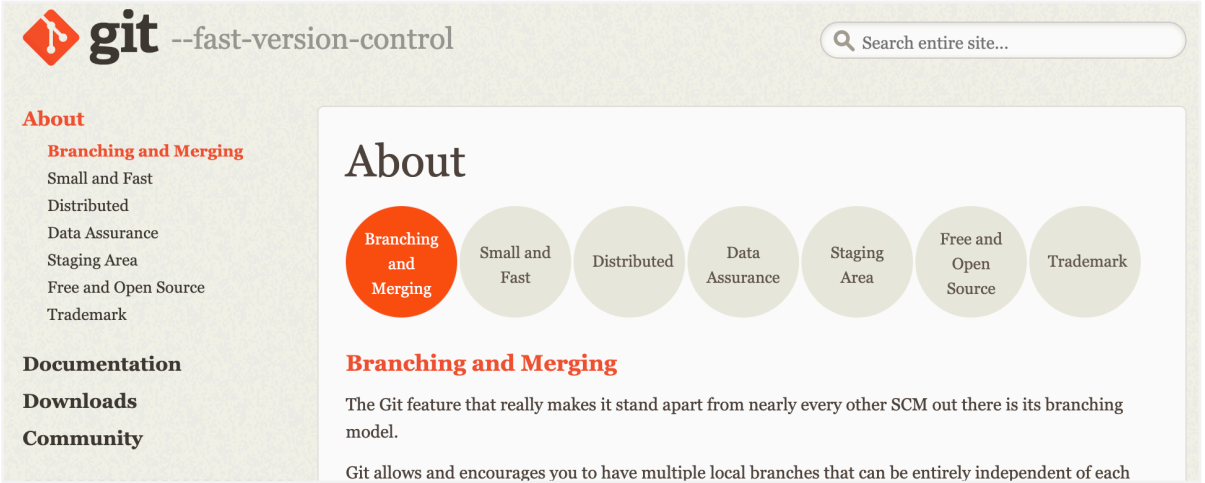
- CVS(Concurrent version system)
  - 1980년대에 만들어진 형상관리 툴이지만 파일 관리나 커밋 중 오류 시 롤백이 되지 않는 등 불편한 문제점이 있어 이후 SVN으로 대체됨
- SVN (subversion)
  - 2000년에 CVS를 대체하기 위해 만들어졌음
  - Trunk, Tag, Branch 구조를 사용
  - 중앙 레파지토리 방식
  - 개발자가 자신만의 version history를 가질 수 없음
- GIT
  - 2005년 리누스 토발즈에 의해 시작
  - 매우 빠른 속도와 분산형 저장소. 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

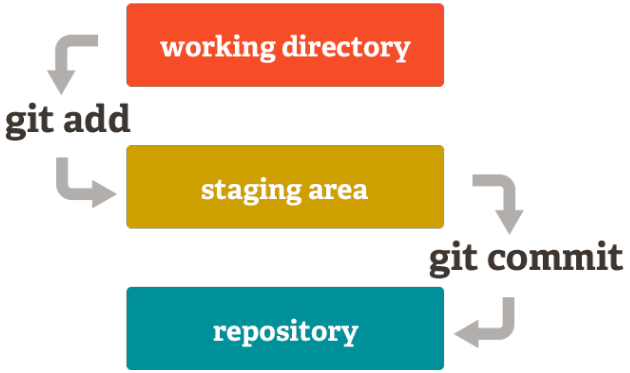


https://git-scm.com/about/

Git 사용해 보기

- github.com 에서 repository 생성
- http://start.spring.io 에서 sample project 생성
- Cloud shell 에 zip 파일 올려서 작업 준비

- git init
- git status
- git add
- git commit
- git remote
- git clone
- git pull
- git push
- Workflowy 에 ‘git 명령어’ 검색



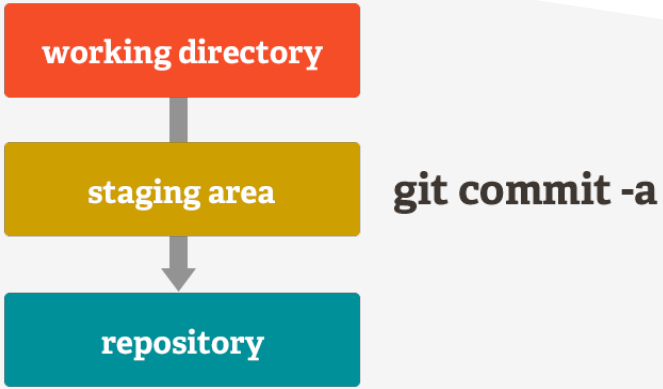


Table of content

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
  - Another Neat Tool
  - 2000 년 출시
  - Base build file : build.xml
  - 유연함이 장점이지만 (모든 명령을 직접작성) , 규칙이 없기때문에 유지보수가 어려움
- Apache Maven
  - Ant 의 불편함을 해소하고자 2004년 출시
  - 규칙을 정하고 Goals 라는 사전 정의된 command 를 제공
  - Base build file : pom.xml
- Gradle
  - 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

" Build Automation "

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>
```

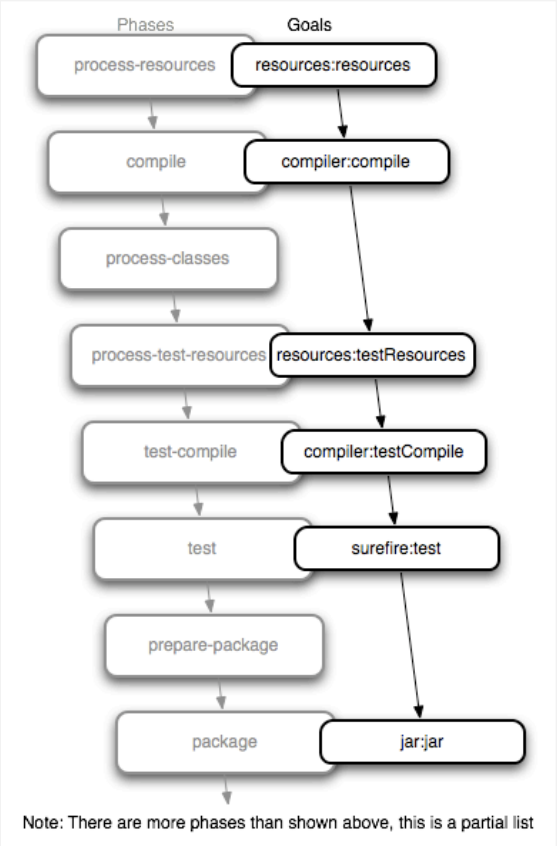
Lifecycle

- 메이븐 프로젝트 생성에 필요한 단계들 (phases) 의 묶음

- default, clean, site 세가지로 표준 정의

- clean : 빌드 시 생성되었던 산출물을 삭제
- site : 프로젝트 문서화 절차
- default : 프로젝트 배포절차, 패키지 타입별로 다르게 정의

- 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 를 해야하니 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