

CI/CD with Azure-pipeline



Copyright © 2020. uEngine-solutions All rights reserved.

What is CI/CD? DevOps? Pipeline?

- CI/CD라는 용어는 애자일 소프트웨어 개발 방법론에서 나옴
- 애자일은 요구사항을 스프린트에 마주어 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/CD를 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 DevOps는 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 CI를 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**

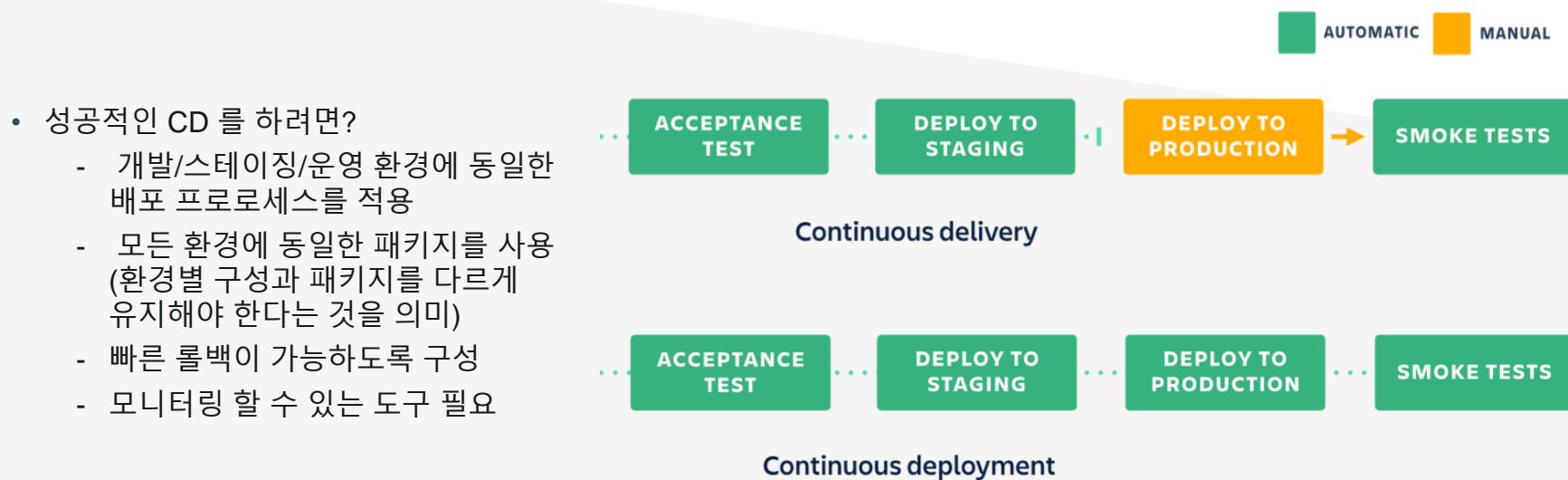


Table of content

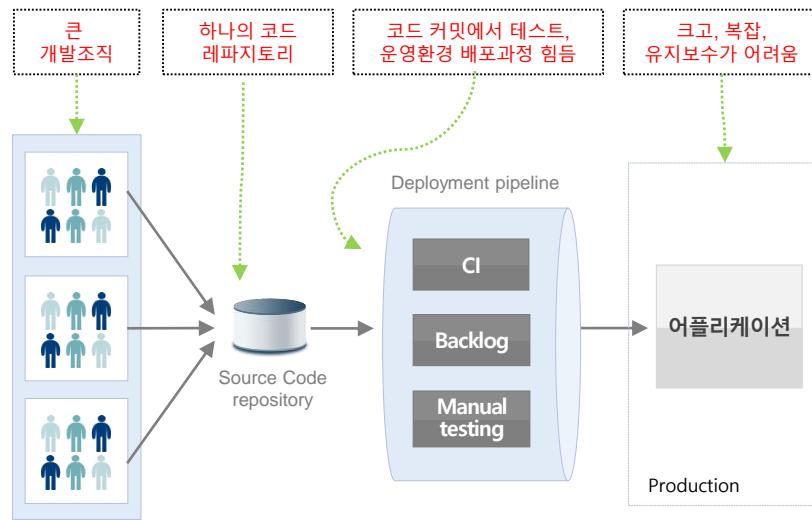
CI/CD with
Azure Pipeline

1. DevOps Process and Tools / Deploy Strategies ✓
2. Version Control , Source Code Management
3. Java build automation tools
4. Azure Devops
5. Azure Pipeline
6. Contract Test
7. Course Test

Process Change : 열차말고 택시를 타라!

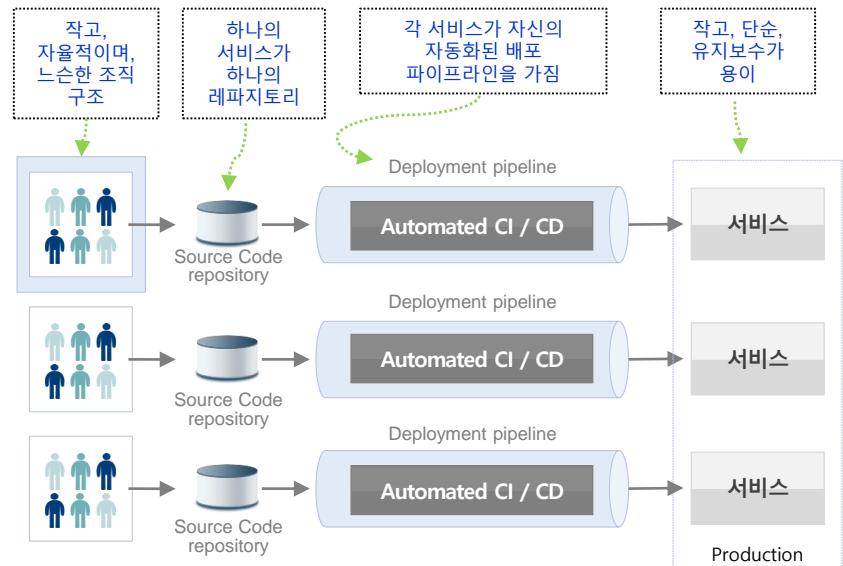
모노리식 개발 및 운영환경

- 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



マイ크로서비스 개발 및 운영환경

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



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

Contract Testing

for Applications with Microservices



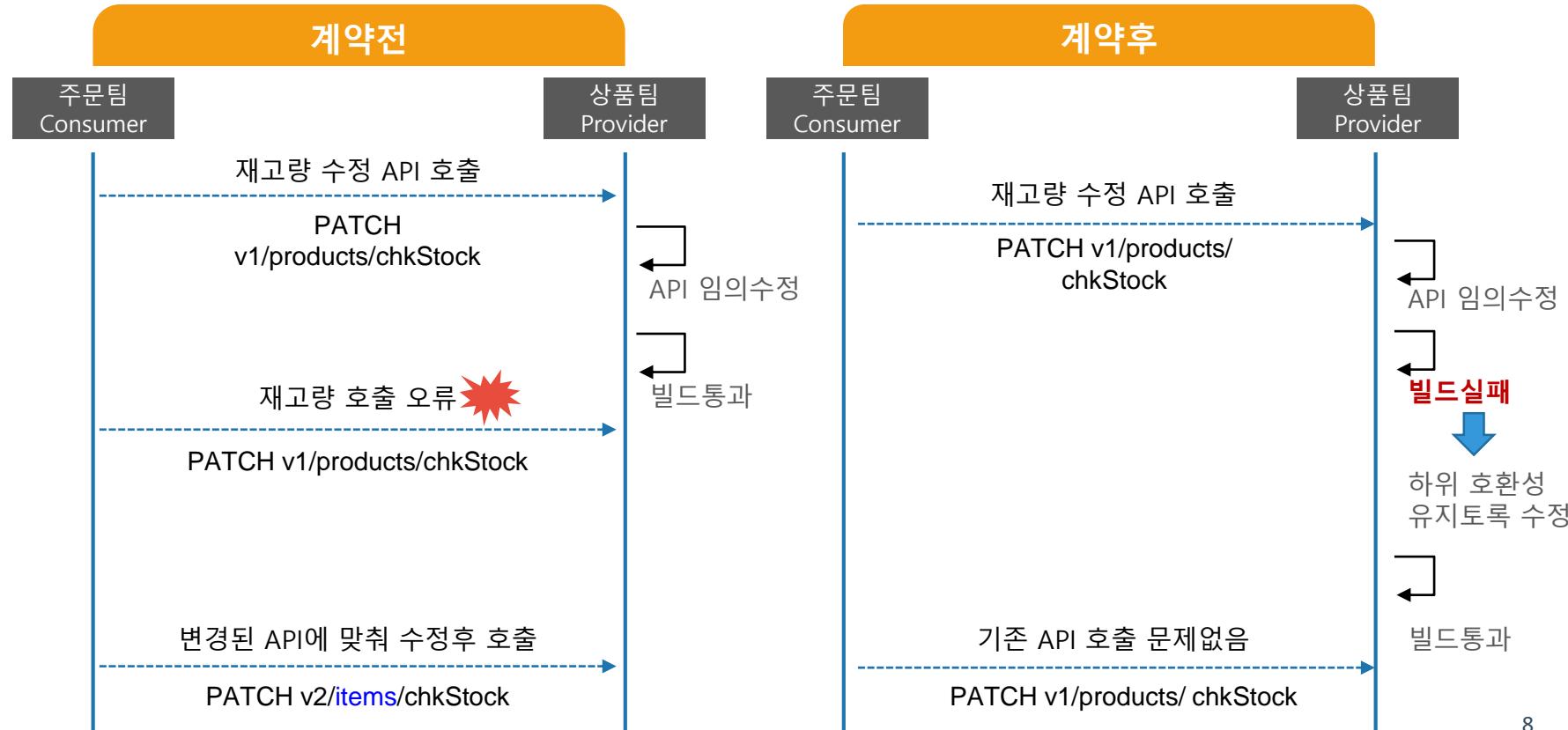
Ex. Spring Cloud Contract, Pact

- 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)

Contract Test



Continuous Delivery

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

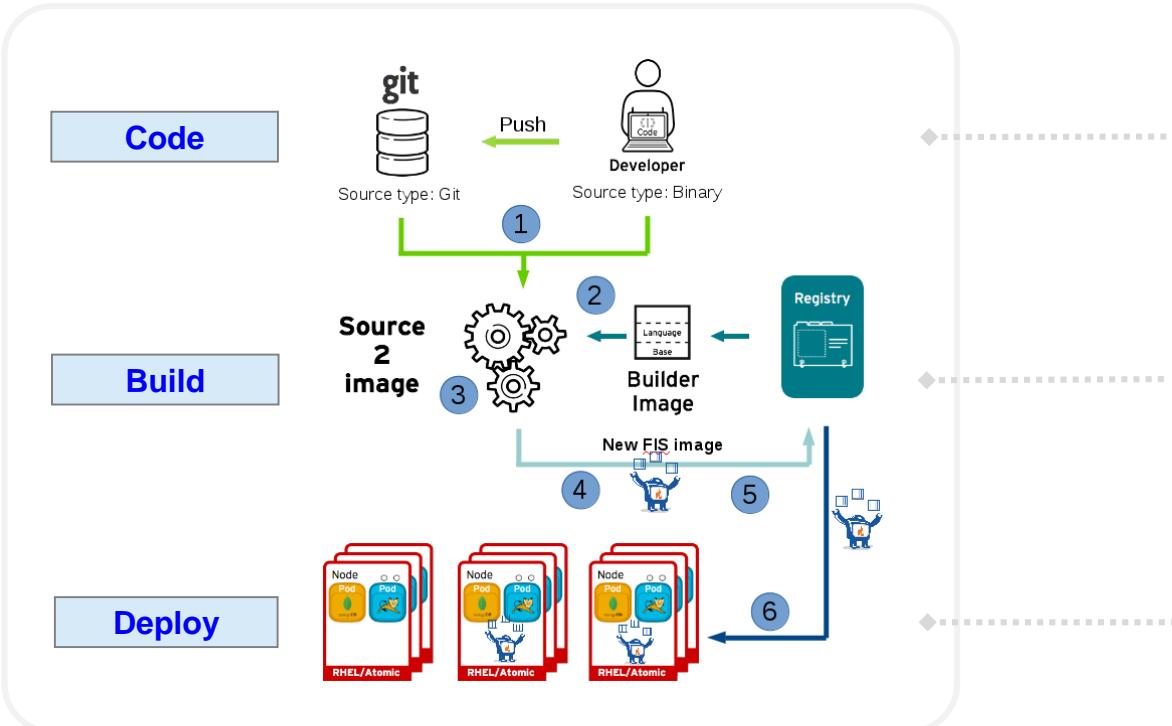


Company	Deploy Frequency	Deploy Lead Time	Reliability	Customer Responsiveness
Amazon	23,000 / day	Minutes	High	High
Google	5,500 / day	Minutes	High	High
Netflix	500 / day	Minutes	High	High
Facebook	1 / day	Hours	High	High
Twitter	3 / week	Hours	High	High
Typical enterprise	Once every 9 months	Months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

출처: 도서 The Phoenix Project

DevOps toolchain

자동화 도구 필요



DevOps platform

컨테이너화 필요

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

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.

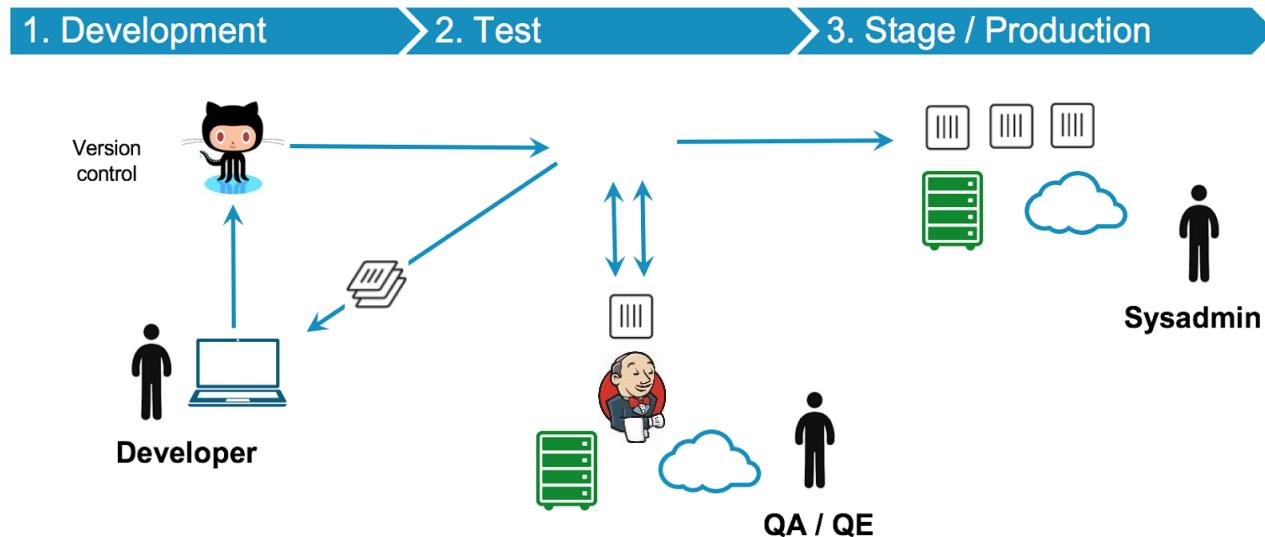
If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.



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	✓	✓	✗	■ ■ ■	□ □ □	□ □ □	■ ■ ■

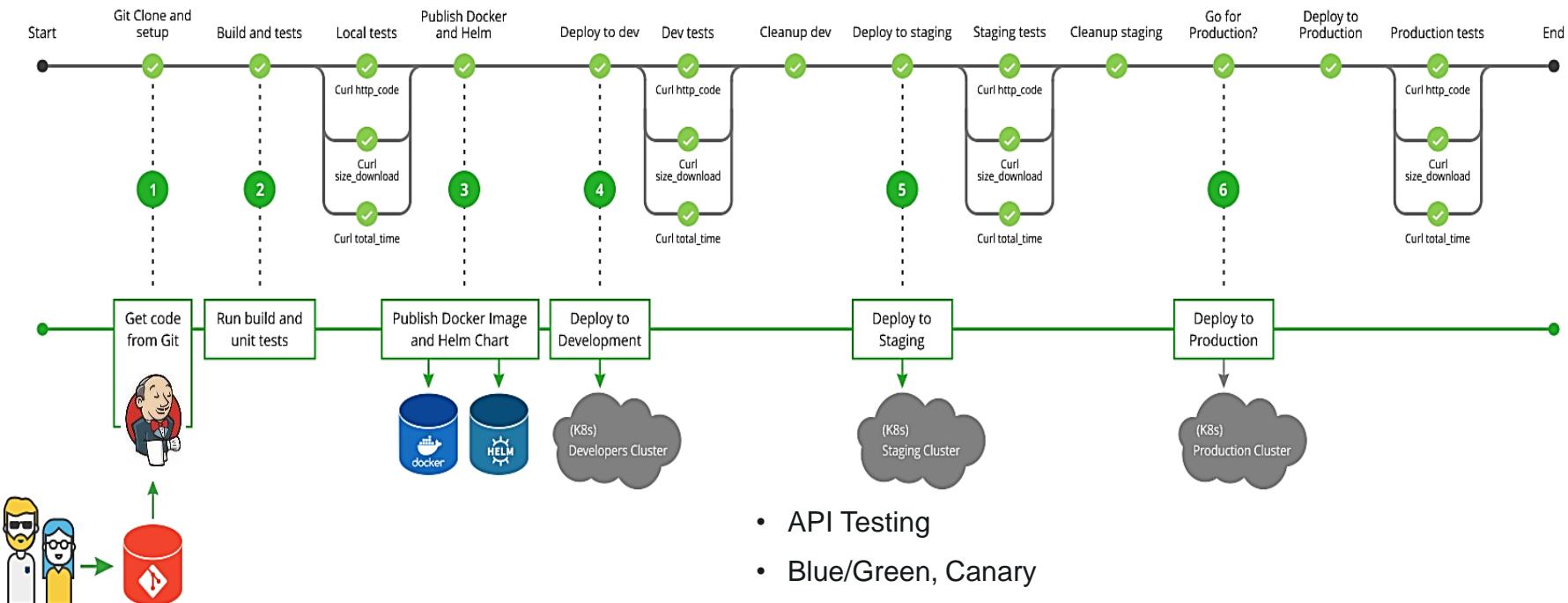
CI/CD Tools – conventional CI/CD

CI /CD Workflow

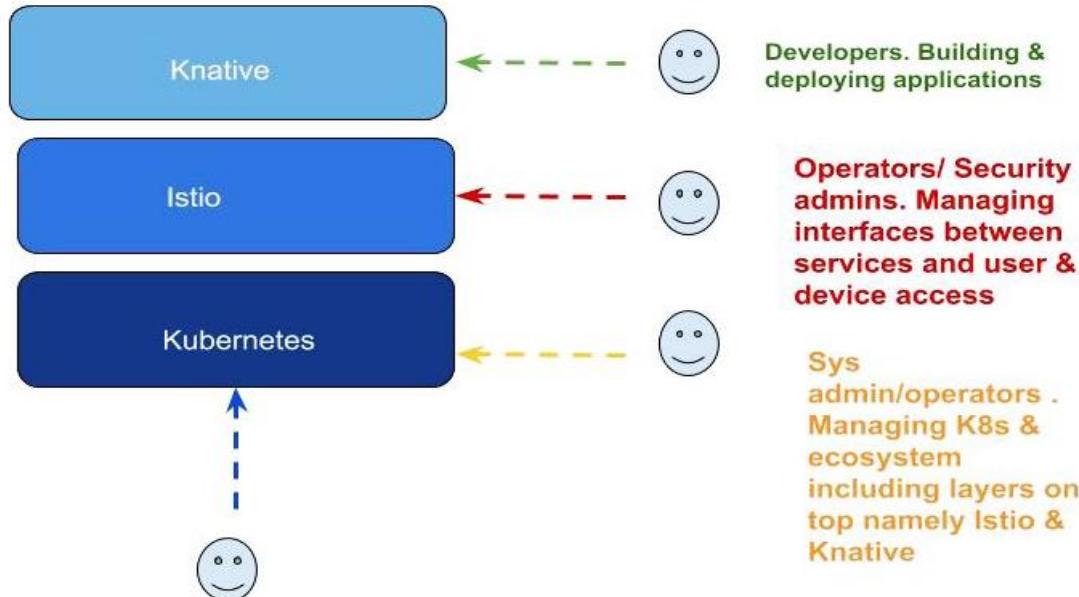


- 잦은 배포
- 일반 서버에 배포
- 테스트 자동화/커버리지
- 도구: Jenkins, Travis, Gitlab 등

CI/CD Tools – Container-based



CI/CD Tools - Serverless



Platform providers or K8s
on-premises(sysadmin/operators)
Making K8s work and interconnect

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

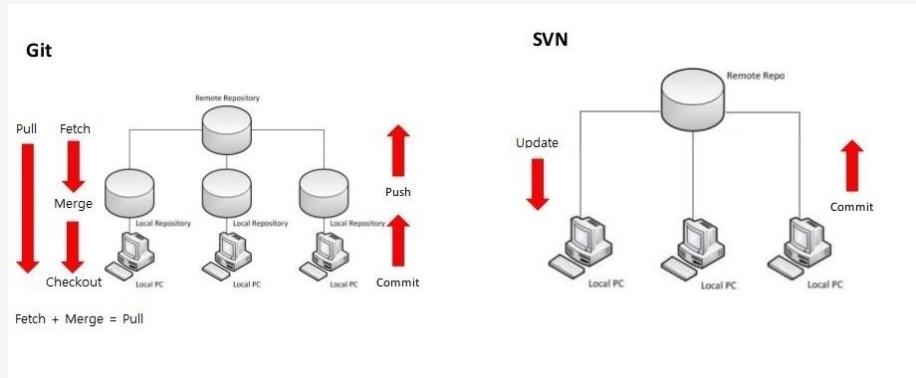
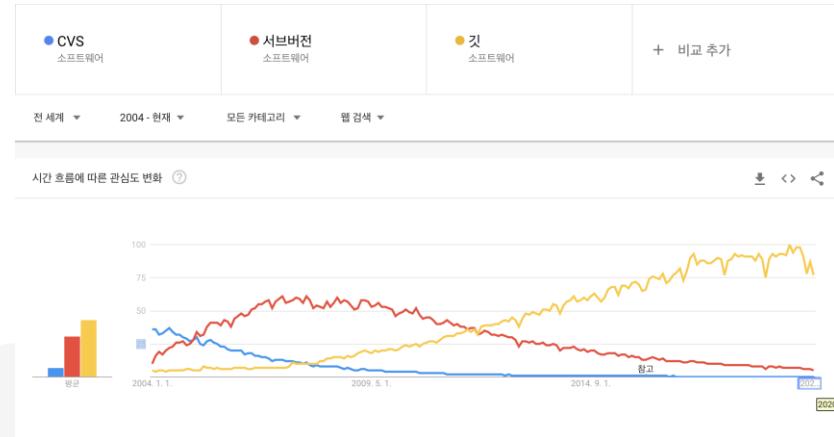
Table of content

CI/CD with
Azure Pipeline

1. DevOps Process and Tools / Deploy Strategies
2. Version Control , Source Code Management ✓
3. Java build automation tools
4. Azure Devops
5. Azure Pipeline
6. Contract Test
7. Course Test

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 리눅스 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 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 'About' section of the official Git website (<https://git-scm.com/about>). The page features a navigation bar with links for 'Documentation', 'Downloads', and 'Community'. A search bar is located in the top right corner. The main content area has a heading 'About' and a list of features: 'Branching and Merging', 'Small and Fast', 'Distributed', 'Data Assurance', 'Staging Area', 'Free and Open Source', and 'Trademark'. The 'Branching and Merging' item is highlighted with a red circle. Below this, there is a section titled 'Branching and Merging' with a descriptive paragraph and a note about the Git feature allowing multiple local branches independent of each other.

<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

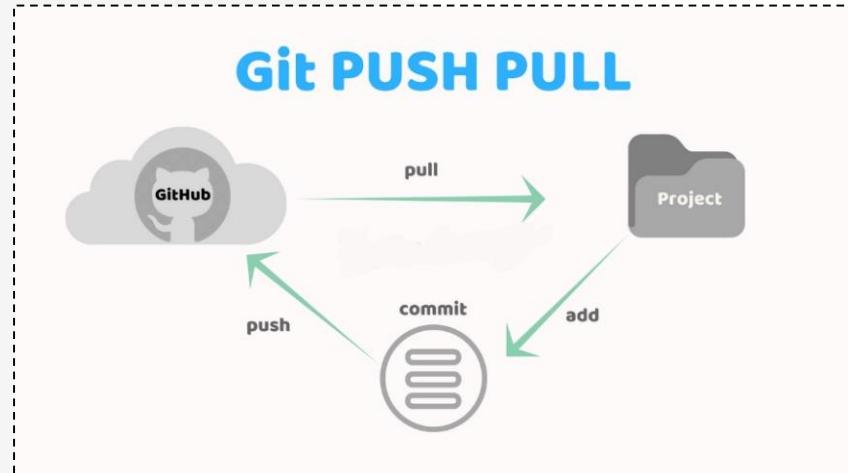
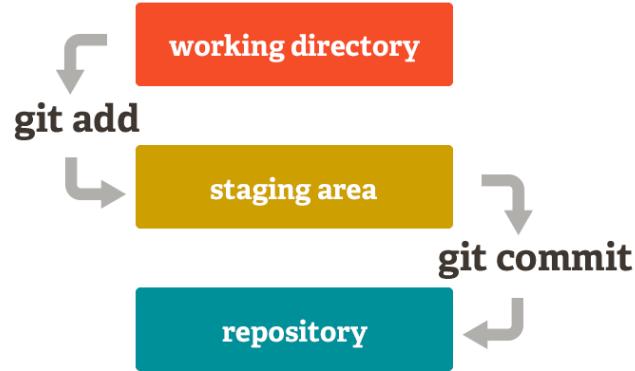


Table of content

CI/CD with
Azure Pipeline

1. DevOps Process and Tools / Deploy Strategies
2. Version Control , Source Code Management
3. Java build automation tools 
4. Azure Devops
5. Azure Pipeline
6. Contract Test
7. Course Test

“

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
 - 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 핵심 개념

- Plugin
- Lifecycle
- Dependency
- Profile
- POM

Plugin

- 메이븐은 플러그인을 구동해주는 프레임워크(plugin execution framework)이다. 모든 작업은 플러그인에서 수행 한다.
- 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>
```

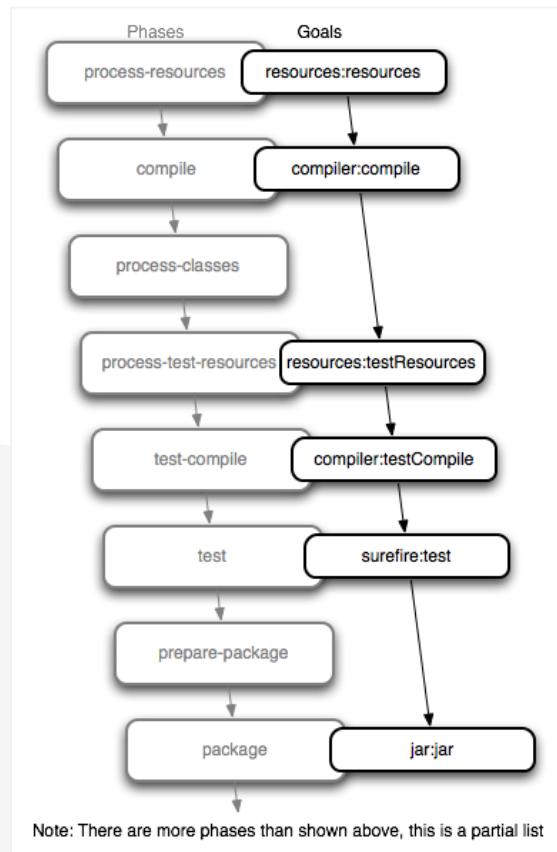
Lifecycle

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 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

Table of content

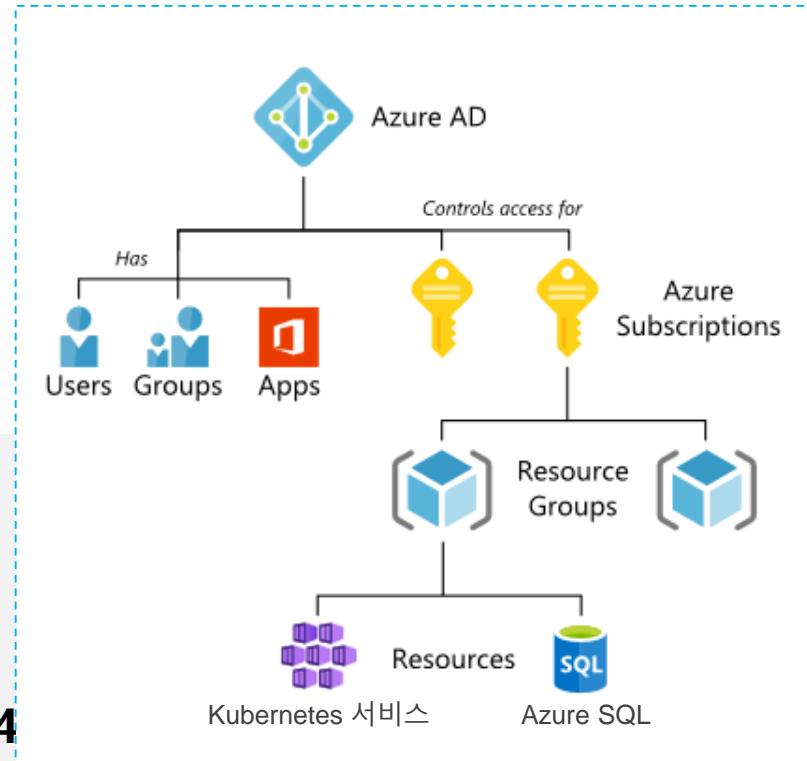
CI/CD with
Azure Pipeline

1. DevOps Process and Tools / Deploy Strategies
2. Version Control , Source Code Management
3. Java build automation tools
4. Azure Devops 
5. Azure Pipeline
6. Contract Test
7. Course Test

Azure Cloud Platform



“가장 많은 글로벌 지역 보유, 56개 리전 14



선행 작업

- 리소스 그룹 생성
- 컨테이너 레지스트리 생성
- AKS 클러스터 생성

리소스 그룹 생성

모든 서비스 > 리소스 그룹 > 리소스 그룹 만들기

리소스 그룹 만들기

기본 태그 검토 + 만들기

리소스 그룹- Azure 솔루션의 관련 리소스를 보관하는 컨테이너입니다. 리소스 그룹에 솔루션의 모든 리소스를 포함할 수도 있고 그룹으로 관리할 리소스만 포함할 수도 있습니다. 무엇이 조직에 가장 적합한지에 따라 리소스 그룹에 리소스를 할당할 방법을 결정합니다. 자세한 정보

프로젝트 정보

구독 * ⓘ

리소스 그룹 * ⓘ

무료 체험

eventstorming

리소스 세부 정보

영역 * ⓘ

(아시아 태평양)대한민국 중부

검토 + 만들기

< 이전

다음: 태그 >

컨테이너 레지스트리 생성

1. 리소스 그룹에서 추가 버튼을 클릭하여 컨테이너 -> Container Registry 를 선택한다.
2. 이름과 위치를 지정하고 생성 버튼을 클릭한다.

모든 서비스 > 리소스 그룹 > eventstorming > 새로 만들기

새로 만들기

Marketplace 검색

Azure Marketplace 모두 보기 추천 모두 보기

- 시작
- 최근에 만들어짐
- AI + 기계 학습
- 분석
- 블록체인
- 계산
- 컨테이너**
- 데이터베이스

1

Container Instances
빠른 시작 자습서

Container Registry
빠른 시작 자습서

Kubernetes Service
빠른 시작 자습서

Web App for Containers
빠른 시작 자습서

모든 서비스 > eventstorming > 새로 만들기 > 컨테이너 레지스트리 만들기

컨테이너 레지스트리 만들기

레지스트리 이름 * **event** .azurecr.io

구독 * 무료 체험

리소스 그룹 * eventstorming

2

새로 만들기

위치 * 대한민국 중부

관리 사용자 * 사용 안 함

SKU * 표준

AKS 클러스터 생성

- 리소스 그룹에서 추가 버튼을 클릭하여 컨테이너 -> Kubernetes Service 를 선택한다.
- 이름과 위치를 지정하고 생성 버튼을 클릭한다.

모든 서비스 > 리소스 그룹 > eventstorming > 새로 만들기

새로 만들기

Marketplace 검색

Azure Marketplace 모두 보기 추천 모두 보기

시작	Container Instances 빠른 시작 자습서
최근에 만들어짐	Container Registry 빠른 시작 자습서
AI + 기계 학습	Kubernetes Service 빠른 시작 자습서
분석	Web App for Containers 빠른 시작 자습서
블록체인	
계산	
컨테이너	
데이터베이스	

모든 서비스 > 리소스 그룹 > eventstorming > 새로 만들기 > Kubernetes 클러스터 만들기

Kubernetes 클러스터 만들기

배포된 리소스와 비용을 관리할 구독을 선택합니다. 풀더 같은 리소스 그룹을 사용하여 모든 리소스를 정리 및 관리합니다.

구독 * ⓘ 무료 체험

리소스 그룹 * ⓘ eventstorming 새로 만들기

클러스터 세부 정보

2 Kubernetes 클러스터 이름 * ⓘ dev1

지역 * ⓘ (아시아 태평양)대한민국 중부

Kubernetes 버전 * ⓘ 1.14.8(기본값)

DNS 이름 접두사 * ⓘ dev1-dns

주 노드 풀

클러스터의 주 노드 풀에 있는 노드의 수와 크기입니다. 프로덕션 워크로드의 경우 복원력을 위해 3개 이상의 노드를 사용하는 것이 좋으며, 개발 또는 테스트 워크로드의 경우 노드가 하나만 필요합니다. 클러스터를 만든 이후에는 노드 크기를 변경할 수 없지만 만든 후 클러스터의 노드 수를 변경할 수 있습니다. 노드 풀을 추가하려면 "크기 조정" 탭에서 "X" 기능을 사용하도록 설정해야 합니다. 그러면 클러스터를 만든 후 노드 풀을 추가할 수 있습니다. [Azure Kubernetes Service의 노드 풀에 대한 자세한 정보](#)

노드 크기 * ⓘ 표준 DS2 v2
크기 변경

노드 개수 * ⓘ 1

검토 + 만들기

클러스터 접속 및 ACR 연결

1. 오른쪽 상단의 cloud shell 연결 버튼을 클릭해서 shell 실행
2. Cloud shell에서 리소스 그룹과 쿠버네티스 클러스터 이름으로 접속
3. az aks get-credentials --resource-group My_Resource_Group --name My-Cluster
4. kubectl get svc 명령어로 접속 확인
5. 클러스터와 ACR 연결
az aks update -n My-Cluster -g My_Resource_Group --attach-acr My-Acr

1

Microsoft Azure 리소스, 서비스 및 문서 검색(G+/)

Azure 서비스

리소스 만들기 모든 리소스 리소스 그룹 가상 머신 App Services 스토리지 계정 SQL 데이터베이스 Azure Database for PostgreSQL Azure

Bash Requesting a Cloud Shell. Succeeded. Connecting terminal...

Welcome to Azure Cloud Shell

Type "az" to use Azure CLI
Type "help" to learn about Cloud Shell

2 sanaloveyou@Azure:~\$ az aks get-credentials --resource-group My_Resource_Group --name My-Cluster
merged "My-Cluster" as current context in /home/sanaloveyou/.kube/config
sanaloveyou@Azure:~\$ kubectl get svc
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
kubernetes ClusterIP 10.0.0.1 <none> 443/TCP 47h
sanaloveyou@Azure:~\$

“

Azure Devops

- Overview
- Board
- Test Plan
- Artifacts

DevOps 프로젝트 생성

1 http://dev.azure.com/ 접속

Azure DevOps
apexacme@uengine.org Switch directory

Get started with Azure DevOps

Choosing Continue means that you agree to our Terms of Service, Privacy Statement, and Code of Conduct.

I would like information, tips, and offers about Azure DevOps and other Microsoft products and services. Privacy Statement.

Continue

2 조직생성

Azure DevOps
apexacme@uengine.org Switch directory

Almost done...

Name your Azure DevOps organization

We'll host your projects in

Continue

Azure DevOps

Plan smarter, collaborate better, and ship faster with a set of modern dev services.

Start free

Start free with GitHub

Already have an account?

[Sign in to Azure DevOps >](#)

3 프로젝트 생성

프로젝트명 입력 >>

Create a project to get started

Project name*

Description

Visibility

Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.

Private
Only people you give access to will be able to view this project.

Advanced



Agile Board

Azure DevOps interface showing the Agile Board backlog for the 'testOps' team. The left sidebar navigation includes: testOps, Overview, Boards, Backlogs (selected), Work items, Boards, Backlogs, Sprints, Queries, Repos, Pipelines, Test Plans, and Artifacts. The main area displays the 'Backlog' tab for the 'testOps Team'. It shows one User Story titled 'CI/CD class'. Filter options include Order, Work Item Type, and Title. A 'New Work Item' button is available.

Azure DevOps interface showing the Agile Board backlog for the 'testOps' team. The left sidebar navigation is identical to the first screenshot. The main area displays the 'Backlog' tab for the 'testOps Team'. A context menu is open over the 'CI/CD class' User Story, listing options: New (selected), Active, Resolved, Closed, and Removed. A large grey arrow points from the first screenshot towards this second one.

Load test

Azure DevOps interface showing the Test Plans > Load test section. The sidebar on the left highlights 'Load test'. A top banner states: "Cloud-based load testing service is deprecated and will not be available after March 31st 2020. Click [here](#) to learn more." The main area shows a search bar and a 'Load tests' section with a 'New' button. Below it is a 'Web Scenarios' table with one entry: 'WebScenario1' containing 'Request1'. To the right, configuration panels show 'HTTP method: GET', 'URL: Enter URL as http://www.your-app...', 'Headers' (empty), and 'QueryString Parameters' (empty).

eventstroming / testOps / Test Plans / Load test

Cloud-based load testing service is deprecated and will not be available after March 31st 2020. Click [here](#) to learn more.

+ New | [Run test](#) | Import .HAR file

Search all load tests, begin typin...

Load tests

All load tests

Web Scenarios [Settings](#)

Save [Run test](#) Import .HAR file

+ Add web scenario + Add URL

WebScenario1	Request1	...

HTTP method: GET URL: Enter URL as http://www.your-app...

+ Add header

QueryString Parameters

Artifacts

The screenshot shows the Azure DevOps interface for managing artifacts. On the left, a sidebar lists various feeds: Overview, Boards, Repos, Pipelines, Test Plans, and Artifacts. The Artifacts feed is currently selected. The main area displays a navigation menu with links to eventstroming/testOps/artifacts/packages, a search bar, and a message: "You can now search for packages across feeds. Try searching for a package in the search box. [Learn more.](#)". Below this is a "Connect to feed" section for "eventstroming". A "NuGet" section lists dotnet, NuGet.exe, Visual Studio, and npm. An "npm" section lists npm. A "Maven" section is highlighted, showing its icon, command line reference link, and a message: "First time using Azure Artifacts with Maven on this machine?". It also includes a "Project setup" section with instructions to add the repository to both pom.xml's <repositories> and <distributionManagement>. Below this is a code snippet for settings.xml:

```
<repository>
  <id>eventstroming</id>
  <url>https://pkgs.dev.azure.com/eventstroming/_packaging/eventstroming/maven</url>
  <releases>
    <enabled>true</enabled>
  </releases>
  <snapshots>
    <enabled>true</enabled>
  </snapshots>
</repository>
```

Further down, there is a section to "Add or edit the settings.xml file in \${user.home}/.m2" with another code snippet:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
  https://maven.apache.org/xsd/settings-1.0.0.xsd">
  <servers>
    <server>
      <id>eventstroming</id>
      <username>eventstroming</username>
      <password>[PERSONAL_ACCESS_TOKEN]</password>
    </server>
  </servers>
</settings>
```

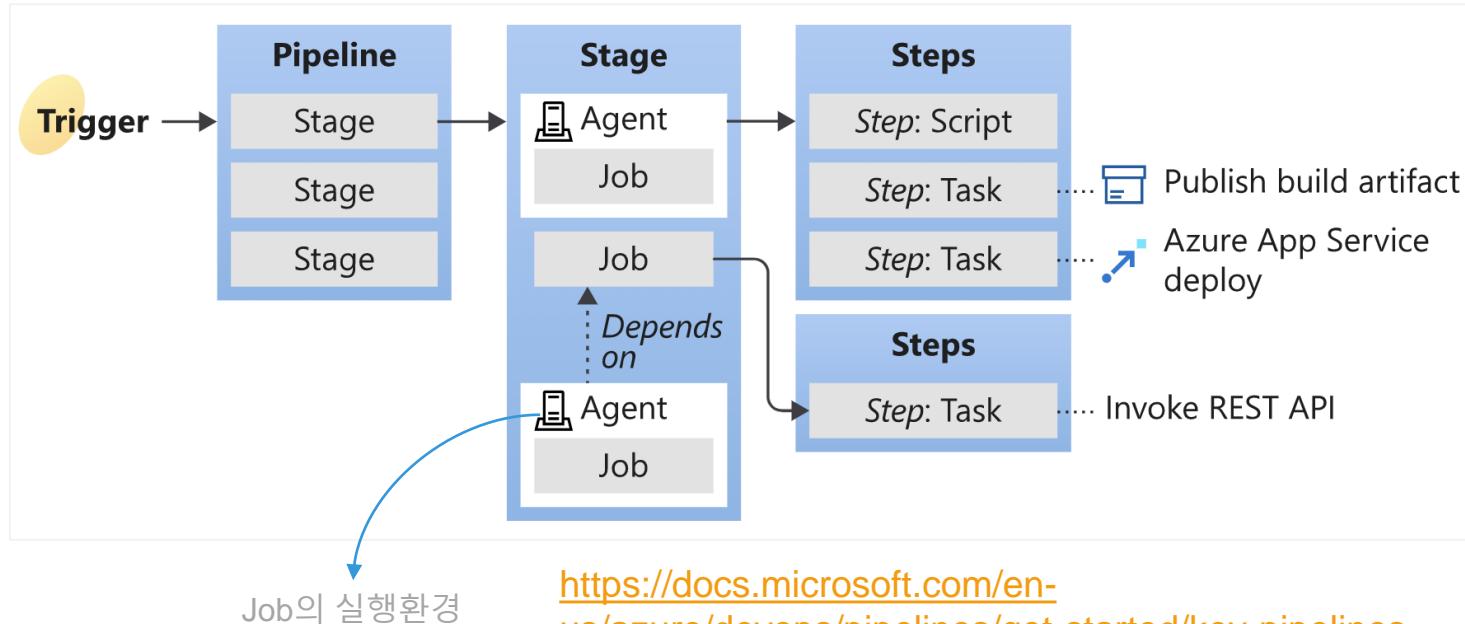
Finally, a note says: "Finally, generate a [personal access token](#) with *Packaging* read & write scope". The bottom section is titled "Restore packages" with the instruction: "Run this command in your project directory" followed by "mvn build".

Table of content

CI/CD with
Azure Pipeline

1. DevOps Process and Tools / Deploy Strategies
2. Version Control , Source Code Management
3. Java build automation tools
4. Azure Devops
5. Azure Pipeline ✓
6. Contract Test
7. Course Test

Azure Pipelines Key Concepts



<https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/key-pipelines-concepts?view=azure-devops>

CI/CD #1 – Dockerizing & API-Based Containerizing

```
git clone https://github.com/<my account>/monolith.git
```

```
cd monolith
```

```
// 소스코드 빌드, 패키징 -> output : jar 파일
```

```
mvn package
```

```
// 도커라이징 -> output : 컨테이너 이미지
```

```
docker build -t (Azure container registry명).azurecr.io/monolith:(tagName) .
```

```
docker push (azure container registry명).azurecr.io/monolith:(tagName)
```

```
// 해당 컨테이너 이미지로 pod 생성
```

```
kubectl create deploy monolith --image= (azure container registry명).azurecr.io/monolith:(tagName)
```

Github 로그인 및 실습 프로젝트 복제

1. <https://github.com/event-storming/monolith> 에 접속
2. 화면 우측의 Fork 버튼 클릭
3. 소스 리파지토리가 자신의 계정으로 복제
- <https://github.com/<my account>/monolith.git>

1

<https://github.com/event-storming/monolith>



The screenshot shows the GitHub repository page for 'event-storming / monolith'. At the top right, there is a 'Fork' button with a red box around it and a red arrow labeled '2' pointing to it. The page displays the following information:

- Repository name: event-storming / monolith
- Issues: 0
- Pull requests: 1
- Actions: 0
- Projects: 0
- Wiki: 0
- Security: 0
- Insights: 0
- Settings: 0
- Commits: 33
- Branches: 1
- Packages: 0
- Releases: 0
- Contributors: 1
- Last commit: 26 days ago by kimscott (feingclient 전환 준비)
- Recent activity:
 - azure 배포 추가 (last month)
 - 오태수정 (3 months ago)
 - 모노리스 feingclient 전환 준비 (26 days ago)
- Branch: master
- New pull request
- Create new file
- Upload files
- Find file
- Clone or download

Azure-pipeline 생성 – CI Classic Editor

The screenshot shows the Azure DevOps interface for creating a new pipeline. A red arrow labeled '1' points to the 'Pipelines' menu item in the sidebar, which is highlighted with a red dashed box. Another red arrow labeled '2' points to the 'Select a source' section where 'GitHub' is selected, also highlighted with a red dashed box.

eventstorming / event-devops / Pipelines

Connect Select Configure Review

New pipeline

Where is your code?

1

2

Select a source

- Azure Repos Git YAML
Free private Git repositories, pull requests, and code search
- Bitbucket Cloud YAML
Hosted by Atlassian
- Github YAML
Home to the world's largest community of developers
- Github Enterprise Server YAML
The self-hosted version of Github Enterprise
- Other Git
Any generic Git repository
- Subversion
Centralized version control by Apache
- Bitbucket Cloud
- Other Git

Authorized using connection: dev1 - GitHub Change ▾

Repository * | Manage on GitHub ↗
kimscott/monolith

Default branch for manual and scheduled builds *
master

Continue

Azure-pipeline 생성 – CI Classic Editor

Empty Job으로 진행

1. Pipeline 의 이름을 지정
2. Pool 선택
3. Agent 를 ubuntu-18.04 선택

The screenshot shows the Azure DevOps CI Classic Editor interface. On the left, there's a pipeline configuration pane with tabs for Tasks, Variables, Triggers, Options, Retention, and History. The Tasks tab is selected. It lists a 'Get sources' task for the 'kimscott/monolith' repository, master branch, and a 'CI' task configured to run on an agent. On the right, the main editor area shows the 'Agent job' configuration for the 'CI' task. A red box highlights the configuration settings:

- Display name ***: CI
- Agent selection ^**
 - Agent pool ⓘ | Pool information | Manage ↗
 - Azure Pipelines
- Agent Specification ***: ubuntu-18.04

At the bottom of the editor, there are 'Demands ⓘ' and other standard editor controls like Save & queue, Discard, Summary, Queue, and View YAML.

Azure-pipeline 생성 – CI Classic Editor

1. + 버튼을 클릭하여 step 을 추가
2. Maven , docker 를 검색하여 각 단계 추가

The screenshot shows the Azure DevOps CI Classic Editor interface. On the left, the pipeline structure is visible:

- Pipeline (Build pipeline)
- Get sources (kimscott/katacoda-scenarios, master)
- ci (Run on agent)
 - + (red box)
 - Maven pom.xml (Maven)
 - buildAndPush (Docker)

On the right, the "Add tasks" search interface is shown. A red box highlights the search bar containing "docker". A red arrow points to the search bar with the number "2". Below the search bar, there are three task cards:

- Docker: Build or push Docker images, login or logout, or run a Docker command. An "Add" button is present.
- Docker CLI installer: Install Docker CLI on agent machine.
- Docker Compose: Build, push, or run multi-container Docker.

Azure-pipeline 생성 – CI Classic Editor

1. Maven 은 package 를 실행할 것이기 때문에 기본으로 놔둔다.

The screenshot shows the Azure DevOps CI Classic Editor interface. On the left, there's a sidebar with tabs: Tasks, Variables, Triggers, Options, Retention, History, Save & queue, Discard, Summary, Queue, and an ellipsis button. Below these tabs, the pipeline structure is listed:

- Pipeline** (Build pipeline)
- Get sources** (kimscott/katacoda-scenarios, master)
- ci** (Run on agent)
 - Maven pom.xml** (Maven) - This task is selected, indicated by a blue checkmark icon. Its configuration panel is open on the right side of the screen.
 - buildAndPush** (Docker)

The configuration panel for the selected Maven task is detailed below:

- Maven** (Task version 3.*):
 - Display name ***: Maven pom.xml
 - Maven POM file ***: pom.xml
 - Goal(s) ***: package
 - Options**

Azure-pipeline 생성 – CI Classic Editor

1. Docker 로 추가한 step 을 선택한다.
2. Container registry 는 이미지를 저장할 저장소를 뜻하는데, 해당 저장소와 연결을 해주어야 한다. New 버튼을 클릭하여 만들어진 저장소를 선택한다.

The screenshot shows the Azure DevOps CI Classic Editor interface. On the left, there's a sidebar with 'Tasks', 'Variables', 'Triggers', 'Options', 'Retention', and 'History' tabs. Below these are sections for 'Pipeline' (Build pipeline), 'Get sources' (from 'kimscott/katacoda-scenarios' repository), and 'ci' (Run on agent). A 'Maven pom.xml' task is listed under 'ci'. The main area displays a 'Docker' task configuration. The 'Task version' dropdown is set to '2.*'. The 'Display name' field contains 'buildAndPush'. Under 'Container Repository', the 'Container registry' dropdown has a red box around it with the number '2'. To the right of the dropdown is a 'Manage' link, a refresh button, and a 'New' button, which is also highlighted with a red box. At the bottom, there are fields for 'Container repository' and 'Commands'.

Azure-pipeline 생성 – CI Classic Editor

1. Azure Container Registry 를 선택한다
2. 컨테이너 레지스트리를 선택하고 Connection name 을 적어준다.

The screenshot shows the Azure DevOps CI Classic Editor interface. On the left, a pipeline named 'Pipeline' is defined with two tasks: 'Get sources' (using 'kimscott/katacoda-scenarios' repository) and 'ci' (Run on agent). The 'ci' task has a 'Maven pom.xml' step and a 'buildAndPush' step (Docker). On the right, a 'New service connection' dialog is open, showing configuration for an 'Azure Container Registry'. The 'Registry type' is set to 'Azure Container Registry' (highlighted with a red box), and the 'Subscription' dropdown shows '종량제1 (1ec8718e-919b-4b61-90a4-7a11a2f8e416)'. The 'Azure container registry' field contains 'user30acr' (highlighted with a red box). In the 'Details' section, the 'Service connection name' is 'acr' (highlighted with a red box), and there is an optional 'Description (optional)' field. Under 'Security', the checkbox 'Grant access permission to all pipelines' is checked. At the bottom right of the dialog is a 'Save' button.

Azure-pipeline 생성 – CI Classic Editor

1. Container repository 명칭은 이미지 명칭이 된다.
2. 아래와 같이 monolith 라는 명칭을 주었을때, 컨테이너 이미지 주소는 [yourAcrName].azurecr.io/monolith 가 된다.

The screenshot shows the Azure DevOps CI Classic Editor interface. On the left, there's a pipeline configuration with several steps: 'Get sources' (GitHub), 'ci' (Run on agent), 'Maven pom.xml' (Maven), and 'buildAndPush' (Docker). The 'buildAndPush' step is currently selected. On the right, the 'Docker' task settings are displayed. The 'Task version' is set to '2.*'. The 'Display name' is 'buildAndPush'. Under the 'Container Repository' section, the 'Container registry' dropdown is set to 'acr', and the 'Container repository' input field is highlighted with a red box and a red arrow pointing to it. Inside the box, the value 'monolith' is shown. Below this, the 'Commands' section shows a 'Command' dropdown set to 'buildAndPush'. At the bottom, the 'Dockerfile' field is present.

Azure-pipeline 생성 – CI Classic Editor

코드 커밋시 자동 트리거 설정

Tasks Variables **Triggers** Options Retention History | Save & queue Discard Summary Queue ...

Continuous integration

kimscott/monolith Enabled

Pull request validation

kimscott/monolith Disabled

Scheduled

No builds scheduled

+ Add

Build completion

Build when another build completes

+ Add

kimscott/monolith

Enable continuous integration

Batch changes while a build is in progress

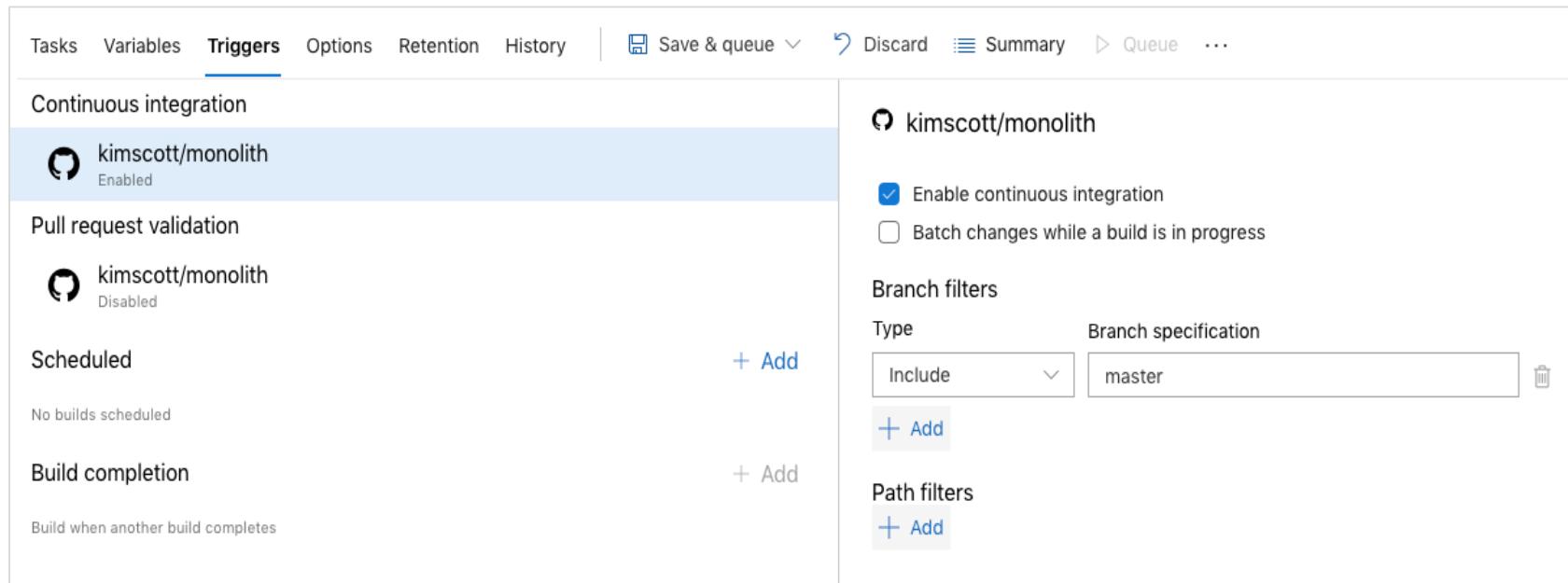
Branch filters

Type	Branch specification
Include	master

+ Add

Path filters

+ Add



Azure-pipeline 생성 – CI Classic Editor

- 파이프라인 상단의 Save & queue > Save 메뉴를 클릭하여 저장한다.
- 파이프라인 리스트 페이지에서, 방금 만든 파이프라인을 선택 한 후, Run pipeline 버튼을 클릭하여 빌드가 정상적으로 되는지 확인한다.

The screenshot shows the Azure DevOps interface for managing pipelines. On the left, the 'Triggers' tab is selected under the 'Continuous integration' section. A red box highlights the 'Save & queue' button, which has a red arrow labeled '1' pointing to it. On the right, a specific pipeline named 'kimscott/monolith' is shown. It has the 'Enable continuous integration' checkbox checked. Below the pipeline details, a recent build run is listed:

Description	Stages	Timestamp
#1 모노리스 feingclient 전환 준비 Manually triggered for master 46419eb	1 stage	Just now 1m 5s

A red box highlights the 'Run pipeline' button in the top right corner of the pipeline card, with a red arrow labeled '2' pointing to it.

Azure-pipeline 생성 – CI Classic Editor

1. 빌드가 정상적으로 끝났을 경우, 리포지토리에 monolith 라는 이름으로 이미지가 생성 된 것을 확인 할 수 있다.

The screenshot shows the Azure DevOps interface for managing a repository named 'momolith'. On the left, there's a sidebar with a search bar and a list of resources. The main area displays the repository details for 'momolith', which was last updated on May 12, 2020, at 1:12 GMT+9. It has one tag and one commit. The repository page includes sections for '리포지토리', 'Webhooks', '복제', and '작업'.

리포지토리	태그 수
momolith	1

태그	마지막 업데이트 날짜	매니페스트 수
1	2020. 5. 12. 오후 1:12 GMT+9	1

Azure-pipeline 생성 – CD Classic Editor

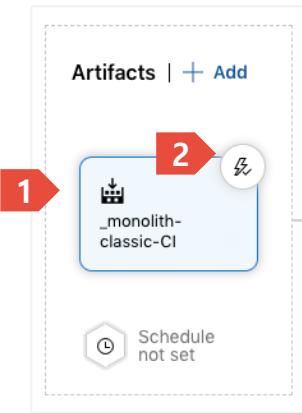
The screenshot shows the Azure DevOps interface for creating a new pipeline. A red arrow labeled '1' points to the 'Releases' button in the left sidebar. A red box highlights the 'Releases' button. A red arrow labeled '2' points to the 'New release pipeline' option in the dropdown menu. A red box highlights the 'New release pipeline' option. A red arrow labeled '3' points to the 'Empty job' template in the 'Select a template' section. A red box highlights the 'Empty job' template.

1. CD 를 생성하기 위하여 Releases Phase를 선택한다.

2. New pipeline 을 클릭하여 새로운 파이프라인을 생성한다.

3. Empty job 을 선택한다.

Azure-pipeline 생성 – CD Classic Editor



1

Add an artifact

Source type

Build Azure Repo... GitHub TFVC

5 more artifact types ▾

Project * event-devops

Source (build pipeline) * monolith-classic-CI

Default version * Latest

Source alias * _monolith-classic-CI

The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of **monolith-classic-CI** published the following artifacts: **Artifact**.

Add

2

Continuous deployment trigger

Build: _monolith-classic-CI

Enabled
Creates a release every time a new build is available.

Build branch filters ⓘ
No filters added.
+ Add | ▾

Pull request trigger

Build: _monolith-classic-CI

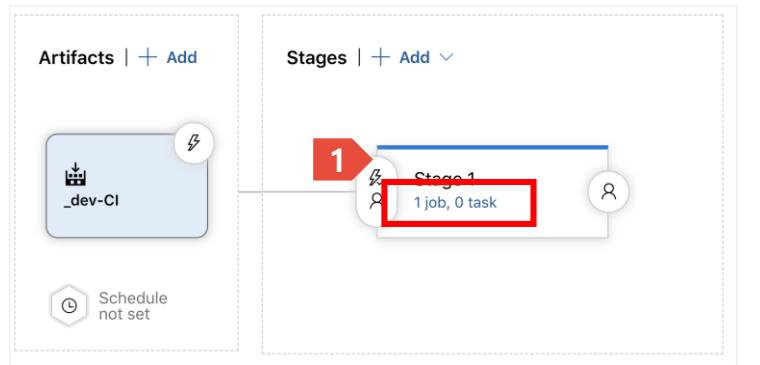
Disabled
Enabling this will create a release every time a selected artifact is available as part of a pull request workflow

1. Artifacts : CI 빌드 선택

2. Trigger Enable 을 선택하여 CI 가 완료 되었을 때, 자동으로 파이프라인이 동작하도록 설정

Azure-pipeline 생성 – CD Classic Editor

1. Stages 의 task 를 선택한다.
2. 첫 화면의 Agent job 에서 파이프라인이 작동할 환경을 ubuntu 18.04 로 설정한다.



The screenshot shows the 'Agent job' configuration page. At the top, there are tabs for Pipeline, Tasks, Variables, Retention, Options, and History. The 'Tasks' tab is selected. Below the tabs, there's a card for 'Stage 1 Deployment process'. Underneath it, there's an 'Agent job' card with the subtext 'Run on agent'. To the right, there's a configuration panel for 'Agent job'. It includes fields for 'Display name *' (set to 'Agent job'), 'Agent selection ^', 'Agent pool' (set to 'Azure Pipelines'), and 'Agent Specification *'. A large red arrow labeled '2' points to the 'Agent Specification' dropdown menu, which is set to 'ubuntu-18.04'.

Azure-pipeline 생성 – CD Classic Editor

1. Agent job 의 + 버튼을 클릭하여 kubectl 을 입력하여 kubectl task 를 추가한다.

The screenshot shows the Azure DevOps CI/CD Classic Editor interface. The top navigation bar includes Pipeline, Tasks (underlined), Variables, Retention, Options, and History. The main area displays a pipeline structure with Stage 1 (Deployment process) containing an Agent job (Run on agent) which has a Kubectl task assigned. On the right, a modal window titled 'Add tasks' is open, featuring a search bar with the term 'kubectl' highlighted by a red box and a red arrow pointing to it. Below the search bar are two task options: 'Kubectl' and 'Kubectl tool installer'. Both options include a brief description and a small icon.

Azure-pipeline 생성 – CD Classic Editor

1. Kubectl로 설정된 task를 선택한 후에 클러스터 연결을 위하여 Kubernetes service connection의 New 버튼을 클릭한다.
2. 클러스터 선택을 하고, 이름을 적은 후에 Save를 클릭하여 저장

Pipeline Tasks Variables Retention Options History

Stage 1 Deployment process

Agent job Run on agent

kubectl Some settings need attention

Kubectl Task version 1* View YAML Remove

Display name * kubectl

Kubernetes Cluster ^

Service connection type * Manage

Kubernetes Service Connection New

This setting is required.

New service connection

Authentication method

- KubeConfig
- Service Account
- Azure Subscription

Azure Subscription

종랑제1 (1ec8718e-919b-4b61-90a4-7a11a2f8e416)

Cluster

team1aks (team1rg)

Namespace

default

Use cluster admin credentials

Details

Service connection name aks

Description (optional)

Security

Grant access permission to all pipelines

Learn more Troubleshoot

Save

Azure-pipeline 생성 – CD Classic Editor

1. Command 에 create 을 선택한다.
2. Arguments 에 deploy monolith --image=[your container registry].azurecr.io/monolith:\${Build.BuildId} 를 입력한다.
3. 해당 명령어는 kubectl create deploy monolith --image=[imageName] 과 같은 역할을 한다.

The screenshot shows the 'Commands' section of the Azure DevOps CD Classic Editor. A 'create' command is selected. The 'Arguments' field contains the command: 'deploy monolith --image=user30acr.azurecr.io/monolith:\${Build.BuildId}'. There is also a 'Use configuration' checkbox and a '...' button.

Commands ^

Command ⓘ

create

Use configuration ⓘ

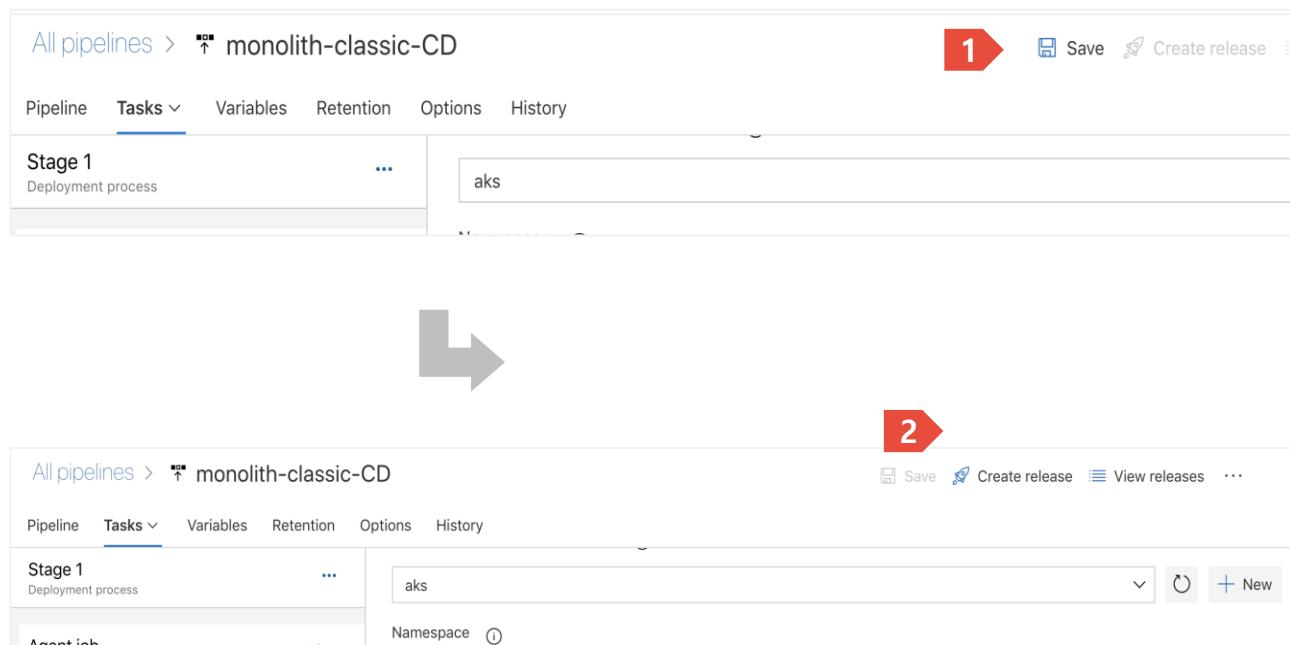
Arguments ⓘ

deploy monolith --image=user30acr.azurecr.io/monolith:\${Build.BuildId}

...

Azure-pipeline 생성 – CD Classic Editor

1. 상단의 Save 버튼을 클릭하여 파이프라인을 저장한다.
2. Create release 버튼을 클릭하여 파이프라인을 실행한다.
3. 파이프라인이 정상적으로 실행시 클러스터에 접속하여 kubectl get po 명령어를 호출하여 monolith pod 가 떠있는지 확인 한다.
4. 확인 후 다시 삭제한다.
kubectl delete deploy monolith



CI/CD #2 – Dockerizing & YAML 기반 CD Pipeline

(YAML - Deployment Descriptor)

```
git clone https://github.com/<my account>/monolith.git  
cd monolith
```

```
// 소스코드 빌드, 패키징 → output : jar 파일  
mvn package
```

```
// 도커라이징 -> output : 컨테이너 이미지  
docker build -t (Azure container registry명).azurecr.io/monolith:(tagName) .  
docker push (azure container registry명).azurecr.io/monolith:(tagName)
```

```
// 배포 관련 yaml 파일을 사용하여 클러스터에 배포  
kubectl apply -f deployment.yaml (Container의 이미지 경로를 각 ACR정보로 수정)  
kubectl apply -f service.yaml
```



```
kubectl set image deployment/monolith monolith=(acr registry명).azurecr.io/monolith:(tagName)
```

CI-Pipeline 변경 : 파이프라인에 YAML 추가

- ‘Maven’ Task 이후에 ‘Copy Files’ , ‘Publish Build Artifact’ 를 추가한다.
- ‘Copy Files’ Task를 선택하여 아래와 같이 설정 값을 지정한다.
- 저장 (Save&queue > Save)

‘Copy Files’ Task 속성설정

name	value
Source Folder	\$(system.defaultworkingdirectory)
Contents	azure/*
Target Folder	\$(build.artifactstagingdirectory)

CI-Pipeline 변경 : 빌드 후 Artifacts 확인

1. 파이프라인을 다시 실행(RUN)하고, 빌드가 성공하면 아래 그림처럼 published 결과 파일이 생성 된다.
2. 해당 링크를 클릭하여 정상적으로 파일을 가져 왔는지 확인 한다.

Summary Releases Code Coverage

Manually run by global korea

Repository and version

event-storming/monolith

master ↗ 46419eb

Time started and elapsed

Today at 오후 5:19

3m 6s

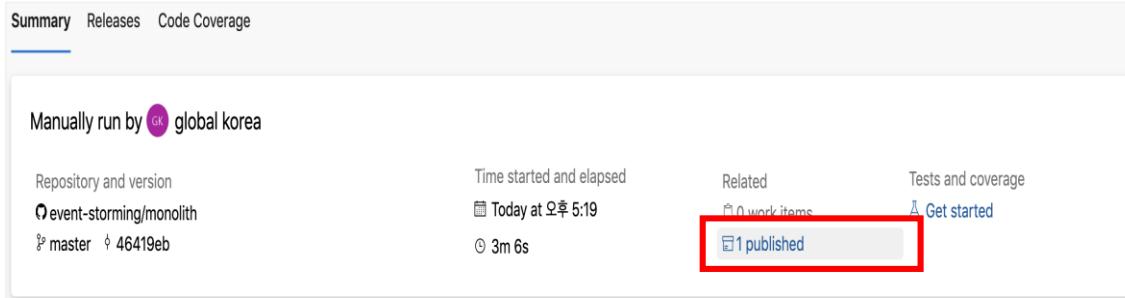
Related

0 work items

Tests and coverage

Get started

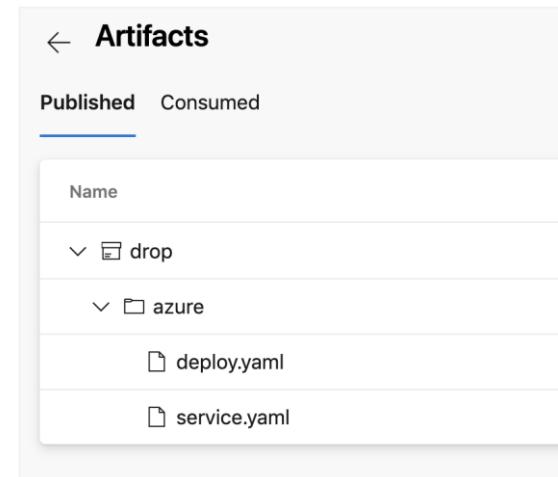
1 published



← Artifacts

Published Consumed

Name
drop
azure
deploy.yaml
service.yaml



CD-Pipeline 변경

Releases 메뉴에서 기존에 만들었던 파이프라인을 선택하고 Edit 버튼을 클릭하여 수정화면으로 이동한다.

The screenshot shows the Azure DevOps interface. On the left, there is a sidebar with the following items:

- dev
- Overview
- Boards
- Repos
- Pipelines (selected)
- Pipelines
- Environments
- Releases (selected)
- Library
- Task groups
- Deployment groups
- Test Plans

The main area shows a search bar at the top with the placeholder "Search all pipelines". Below it, there is a list of pipelines:

- + New
- monolith-classic-CD (highlighted with a red box)

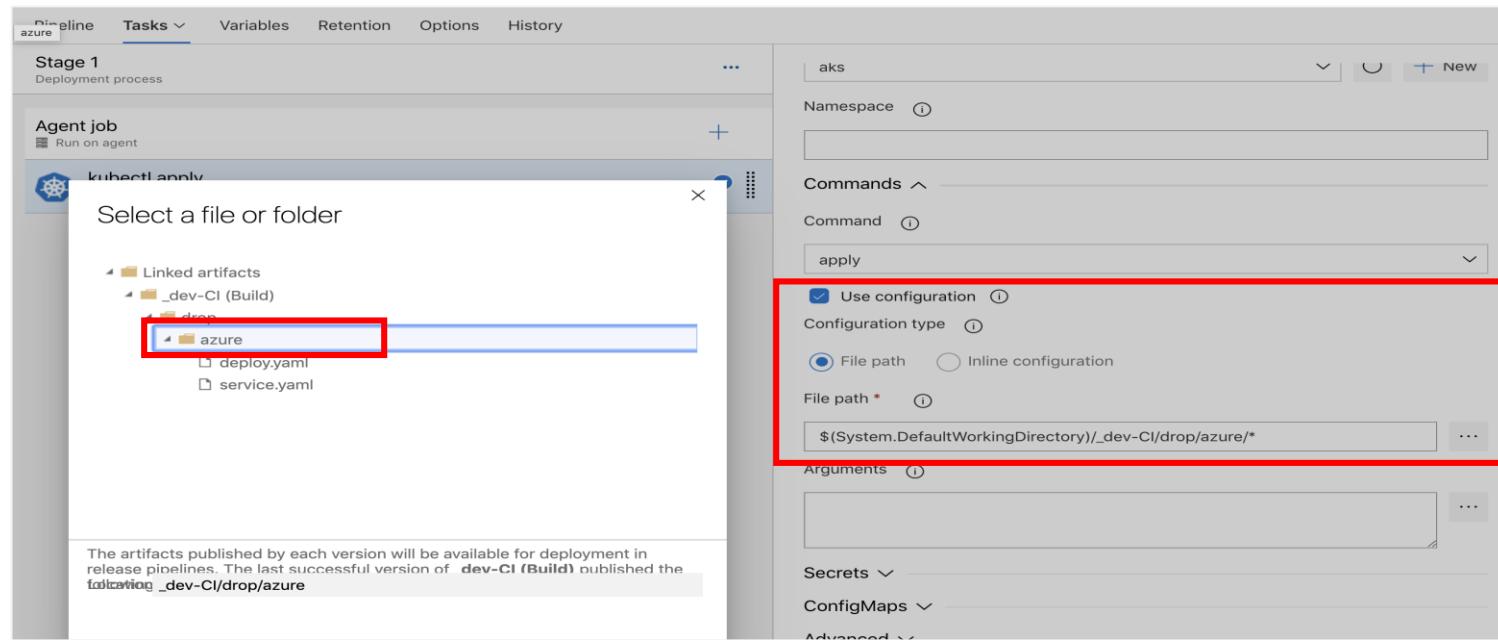
The pipeline details page for "monolith-classic-CD" is shown on the right. The page has tabs for Releases, Deployments, and Analytics. The Releases tab is selected. It displays the following content:

- No releases found
- A cartoon illustration of a person and a dog launching a rocket.
- No releases found
- You can create a new release manually or [setup triggers](#) to create it automatically
- Create a release

The "Edit" button in the top right corner of the pipeline details page is also highlighted with a red box.

CD-Pipeline 변경 : YAML-based KUBECTL

- 앞서 생성한 Release의 kubectl Task에서 'create' 명령어를 'apply'로 변경한다.
- Command 명령어 하단의 Use configuration 체크박스를 체크하면, 파일을 선택할 수 있는 팝업이 열린다.
- CI 결과물에서 추가된 azure 폴더 하위의 첫번째 파일을 선택(deploy.yaml) 한다.
(이때, Arguments 필드는 공백으로 기존 정보를 삭제한다.)
- Kubectl task를 하나 더 추가하여, 위의 2~3번 반복(service.yaml 파일 선택) 한다.



CD-Pipeline 변경 : YAML-based KUBECTL

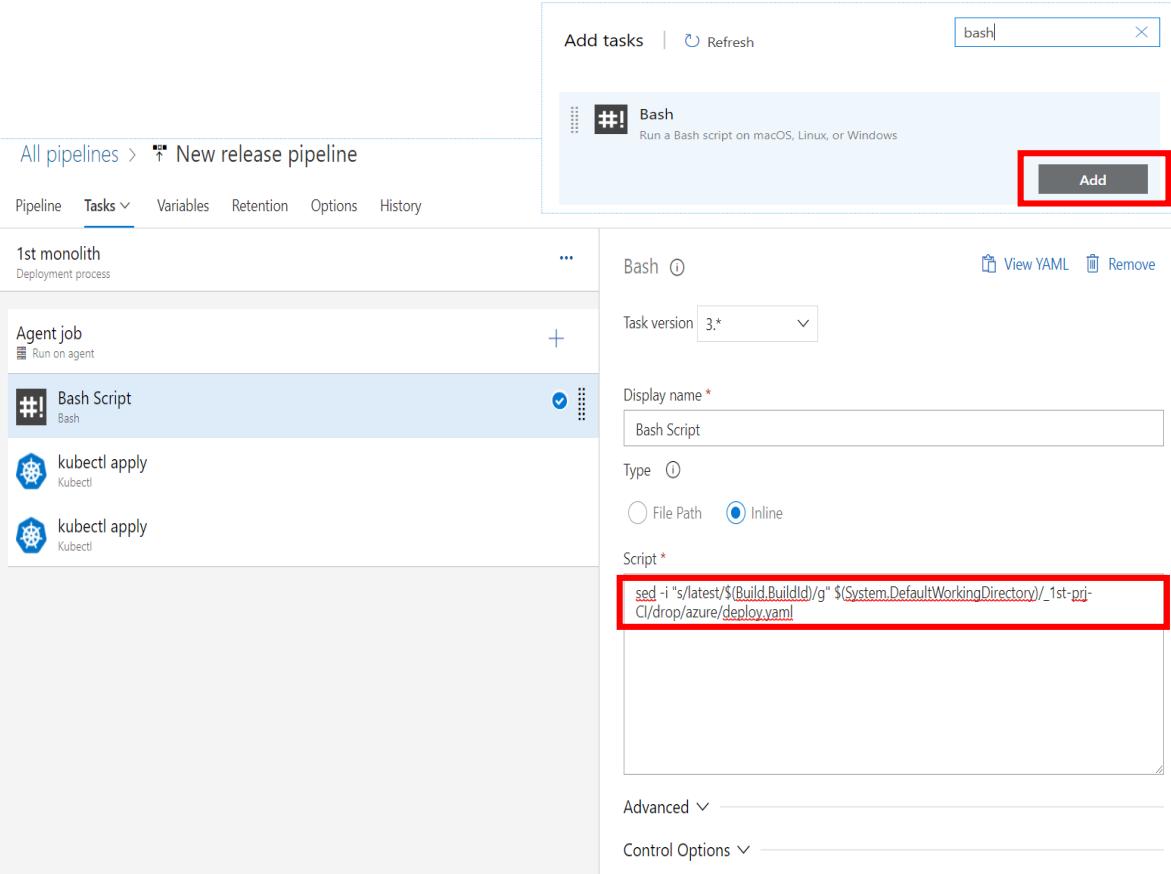
5. 이전 화면의 설정에서 이제 `deploy.yaml` 파일과 `service.yaml` 파일이 `kubectl apply -f` 명령으로 실행된다.

The screenshot shows the 'Pipeline' tab selected in the Azure DevOps 'New release pipeline' interface. A 'kubernetes' task is highlighted, indicating it is currently being edited. The task configuration includes:

- Kubectl**: Task version 1.*
- Display name**: kubectl apply
- Kubernetes Cluster**: Kubernetes Service Connection (aks1)
- Service connection type**: Kubernetes Service Connection
- Kubernetes service connection**: aks1
- Namespace**: default
- Commands**: Command: apply

CD-Pipeline 변경 : Apply Azure BuildID

6. Rollout되는 디플로이 객체는 CI 단계에서 생성된 Azure 빌드 ID를 승계해야 하므로 deploy.yaml 파일의 컨테이너 스펙(Spec)에 이를 Injection 해 주어야 한다.
7. 먼저, monolith GitHub 리파지토리에서 deploy.yaml 파일을 편집하여 이미지에 'latest' 태그를 붙여준다.
8. Release pipeline에서 'Bash' Task를 추가하여 모든 'kubectl' Task 앞에 배치한다.
9. 'Bash' Task의 Script 속성에 내용을 추가한다.
(설명: latest 태그를 Azure가 생성한 빌드ID로 치환하는데, 'Kubectl' Task에서 지정한 deploy.yaml 파일 Path를 가리켜야 함)



The screenshot shows the Azure DevOps Pipeline Editor interface. On the left, under 'All pipelines > New release pipeline', there is a list of tasks: '1st monolith Deployment process', 'Agent job Run on agent', 'Bash Script Bash', 'kubectl apply Kubectl', and another 'kubectl apply Kubectl'. A red box highlights the 'Add' button in the top right corner of the task list area. On the right, a detailed view of the 'Bash Script' task is shown. It has a 'Display name *' of 'Bash Script', is set to 'Type: Inline', and its 'Script *' field contains the command: `sed -i "s/latest/${Build.BuildId}/g" ${System.DefaultWorkingDirectory}/_1st-prj-Cl/drop/azure/deploy.yaml`. This script is intended to replace the 'latest' tag in the Kubernetes deployment YAML file with the current Azure build ID.

Azure-pipeline CI/CD #2 실행 확인

수정한 내용이 반영되도록 저장하고, Github의 monolith 레파지토리의 README.md 파일을 수정하여 정상적으로 CI/CD 파이프 라인을 통해 Kubernetes Cluster에 반영 되는지 확인

- README.md 파일 수정 후,
- Pipeline(CI) Triggering 실행 확인
- published Artifacts, ACR(Azure Container Registry) image 확인
- Releases(CD) Triggering 실행 확인
- Kubernetes Cluster 배포 확인 (kubectl get all)



Hands On with another Project

https://github.com/event-storming/reqres_delivery 프로젝트 fork 후 delivery라는 이름으로 배포 해주세요.

주의사항

- 해당 프로젝트에는 azure라는 배포 관련 폴더가 없습니다.
- 배포 yaml 파일은 kubernetes라는 폴더에 있습니다.
- (**Copy files**에서 경로 수정 주의)
- (CD 생성 시, CI에서 나온 파이프라인을 잘 선택해야 함.)

- 위에서 배우신 set image 명령어를 자세히 살펴보고 변경을 하셔야 합니다.
- `kubectl set image deployment/monolith monolith=[your-acr].azurecr.io/monolith:${(Build.BuildId)}`
- 위 명령어는 deployment/monolith (monolith로 배포된 deployment)를 찾아서 그중 monolith라고 명시된 컨테이너의 image를 변경한다는 의미입니다.

- 배포 완료 후 꼭 클러스터에서 `kubectl get pods` 명령어로 이미지가 정상적으로 running 상태인지 확인해야 합니다.

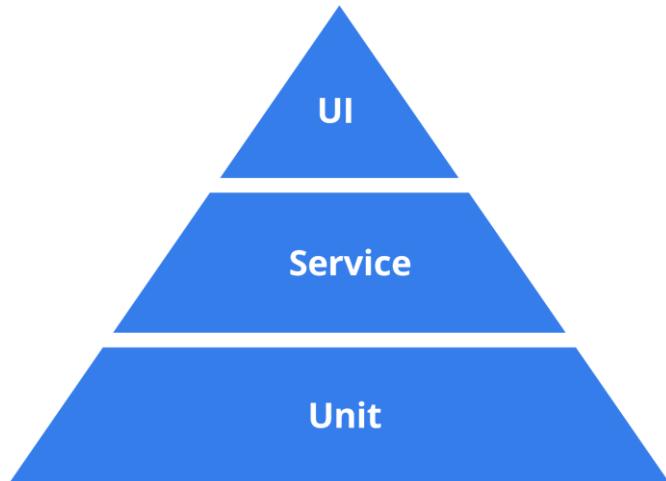
Table of content

CI/CD with
Azure Pipeline

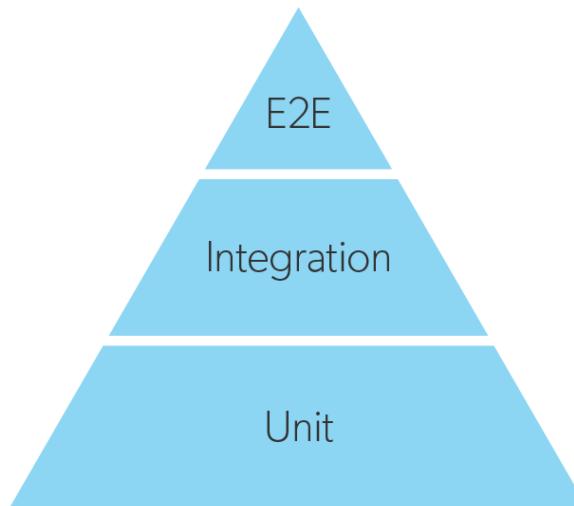
1. DevOps Process and Tools / Deploy Strategies
2. Version Control , Source Code Management
3. Java build automation tools
4. Azure Devops
5. Azure Pipeline
6. Contract Test ✓
7. Course Test

TEST 전략

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



기준 – 테스트 대상



기준 – 테스트 범위

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 빌드하는 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

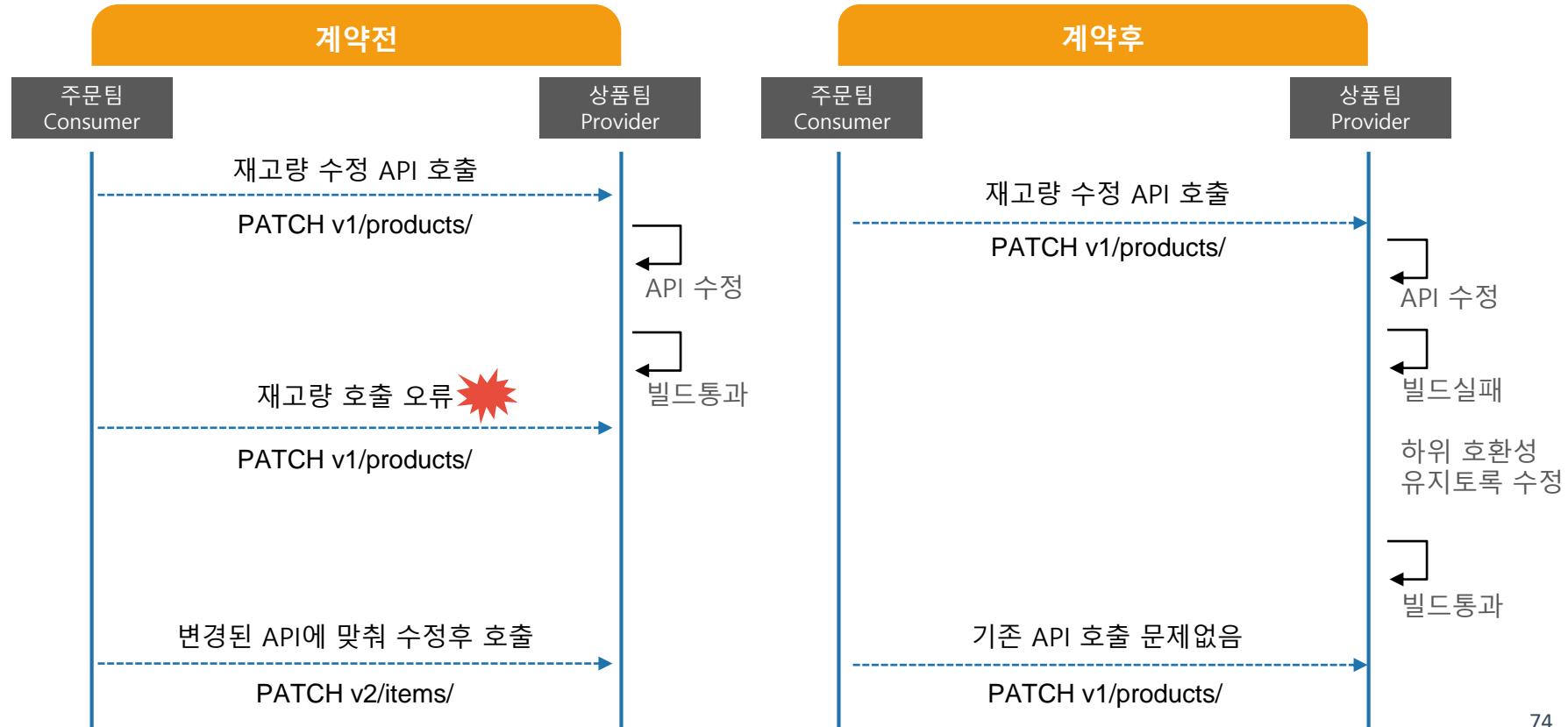
2. 통합 테스트 : 소프트웨어 구성 요소를 함께 테스트

- > 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



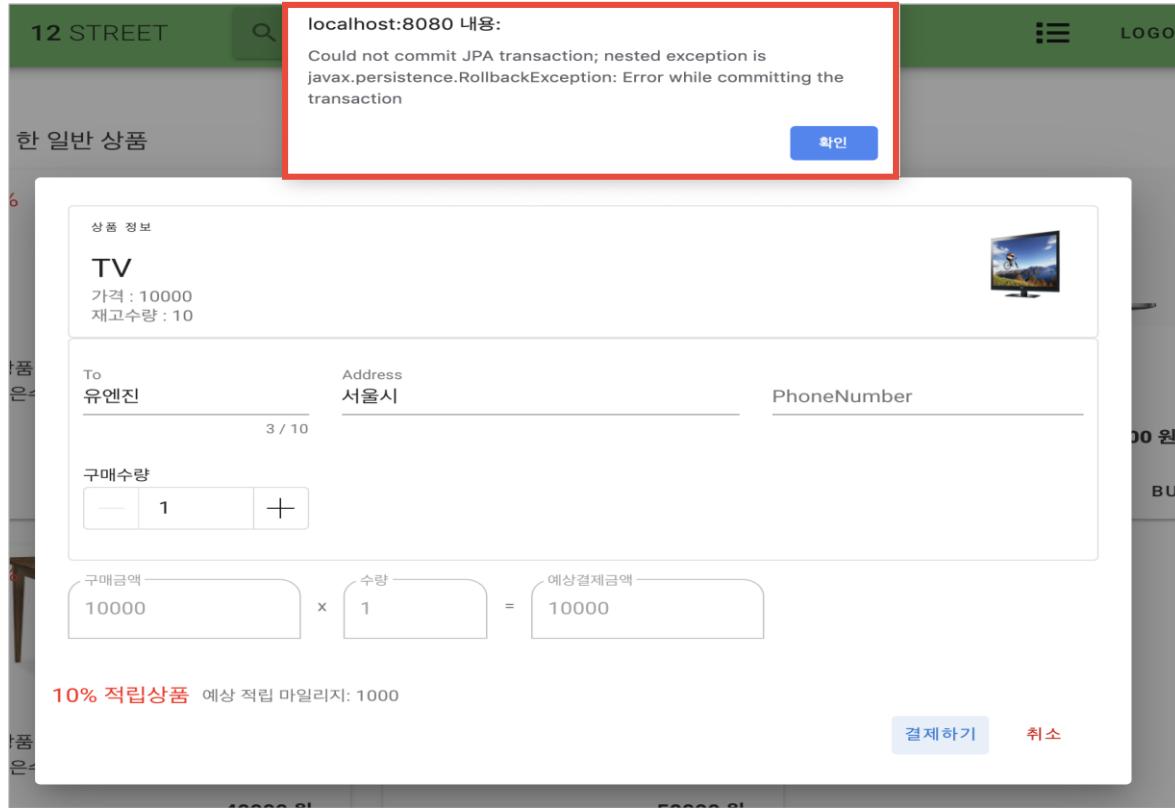
상품팀에서 API를 일방적으로 변경

소스 위치

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

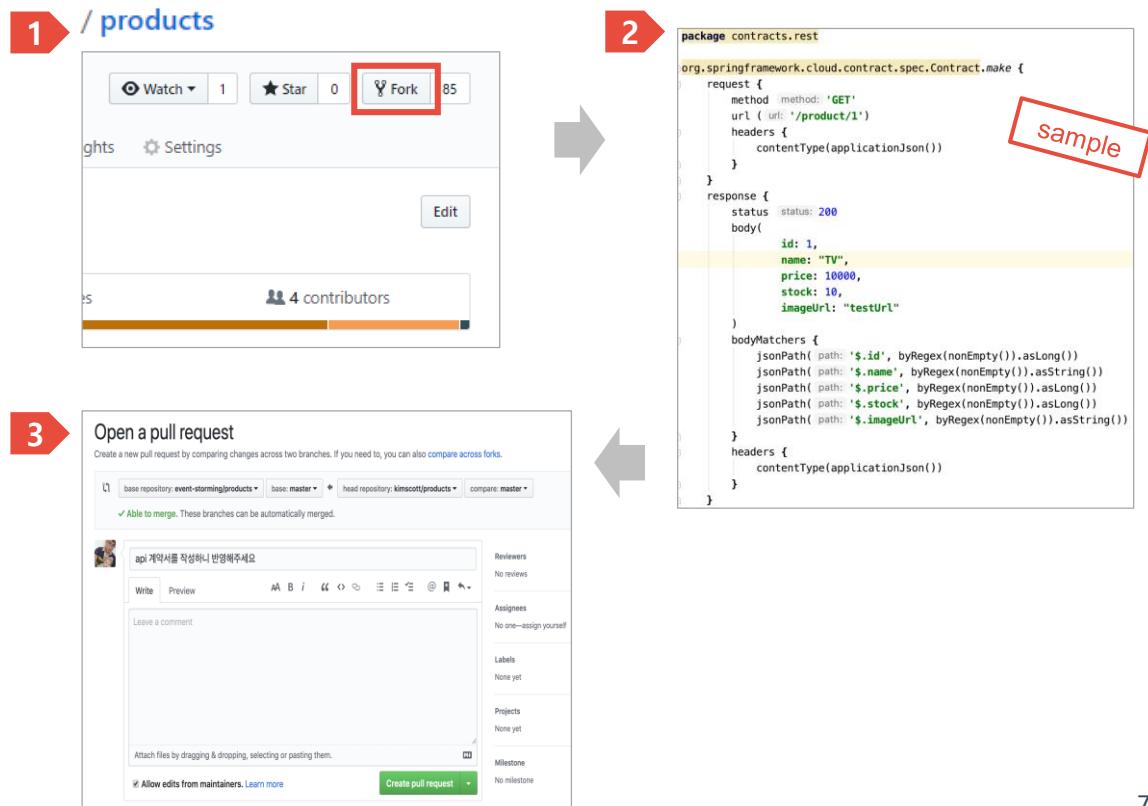
```
13  
14     @GetMapping("/product/{productId}") ← /item/{productId} 로 변경  
15     Product productStockCheck(@PathVariable(value = "productId") Long productId) {  
16         return this.productService.getProductById(productId);  
17     }  
18 }  
19 }
```

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

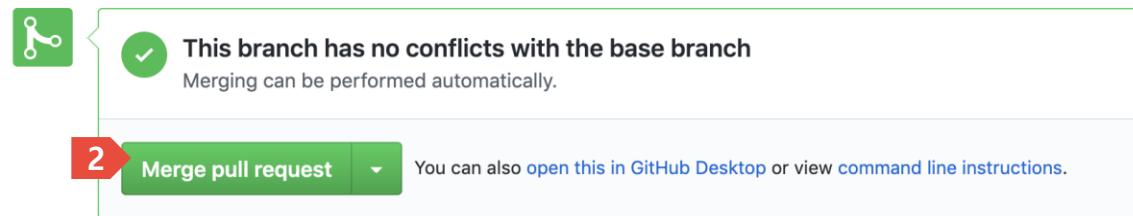
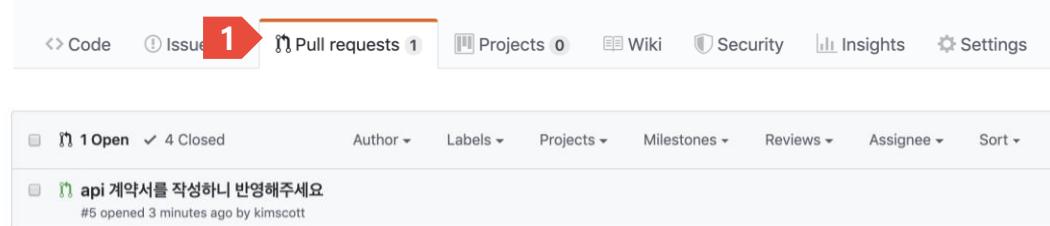
```
package contracts.rest

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



계약체결 WORKAROUND

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

The image shows two screenshots of a GitHub repository interface, likely for a Spring Cloud Contract project.

Screenshot 1 (Top): A screenshot of a GitHub repository page for the branch `master`. The URL is `orders / productGet.groovy`. A red arrow labeled **1** points to the URL bar. The commit details show a change by `kimscott` that adds a file named `productGet.groovy`. The commit message is "계약서 파일". It has 1 contributor and 31 lines (30 sloc) of code. The first few lines of the code are:

```
1 package contracts.rest  
2  
3 org.springframework.cloud.contract.spec.Contract.make {
```

Screenshot 2 (Bottom): A screenshot of the same GitHub repository interface, showing the process of creating a new file. A red arrow labeled **2** points to the URL bar, which now shows the path `products / src / test / resources / contracts / rest / productGet.groovy`. A modal window is open, titled "Edit new file", with the file content area containing the number "1". The "Preview" tab is selected. At the bottom right, there are settings for "Spaces", "2", and "No wrap".

계약체결 후, 상품팀은 계약 위반으로 배포 실패함

Azure-pipeline에서 mvn package 단계 실패

The screenshot shows the Azure Pipeline interface. On the left, under 'Errors', there is one entry for 'Maven' with a status of 'Build failed'. Below this, under 'Jobs', there is a table with one row. The row has a red border around the first column, which contains the name 'Job'. The status is 'Failed' and the duration is '47s'. A large grey arrow points from this screenshot to the detailed log on the right.

Name	Status	Duration
Job	Failed	47s

The screenshot shows the detailed log for the failed 'Maven' step in the pipeline run #2020020. The log output is as follows:

```
609 [INFO]
610 [ERROR] Failures:
611 [ERROR] RestTest.validate_productGet:33
Expecting:
<404>
to be equal to:
<200>
but was not.
[INFO]
618 [ERROR] Tests run: 2, Failures: 1, Errors: 0, Skipped: 0
[INFO]
[INFO] BUILD FAILURE
[INFO]
623 [INFO] Total time: 25.235 s
624 [INFO] Finished at: 2020-02-03T08:06:52Z
[INFO]
626 [ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.22.1:test (defa
[ERROR]
628 [ERROR] Please refer to /home/vsts/work/1/s/target/surefire-reports for the individual test res
629 [ERROR] Please refer to dump files (if any exist) [date].dump, [date]-jvmRun[N].dump and [date]
630 [ERROR] -> [Help 1]
[ERROR]
631 [ERROR]
632 [ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
633 [ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
635 [ERROR] For more information about the errors and possible solutions, please read the following
636 [ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
637 The process '/usr/share/apache-maven-3.6.3/bin/mvn' failed with exit code 1
638 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

Table of content

CI/CD with
Azure Pipeline

1. DevOps Process and Tools / Deploy Strategies
2. Version Control , Source Code Management
3. Java build automation tools
4. Azure Devops
5. Azure Pipeline
6. Contract Test
7. Course Test 