

# THE IaC

# 목적

부트캠프 목적

# 부트캠프 목적

현재 많은 오픈소스가 활성화가 되어 있고, 이를 기반으로 많은 회사들이 제품을 구성 및 판매를 하고 있다. 이와 같은 이유로 점점 리눅스 엔지니어(인프라/개발 구별 없이) 어떠한 솔루션을 어떻게 활용해야 되는지 고민이 많다.

인프라/개발에서 점점 중요하게 여기는 IaC를 어떻게 활용하는지에 대해서 논의한다.

이 부트캠프는 아래와 같은 기술 및 구성으로 작성이 되어있다.

1. 앤서블(추후 Salt/OpenTofu 추가) 기반
2. 쿠버네티스
3. 로키 리눅스(9.x)
4. 오픈스택 2025.1

부트캠프에 현재 프로비저닝 부분은 제외가 되어 있다.

# 랩 환경

랩은 오픈스택 기반으로 제공. 아래 정보로 서버에 접근이 가능하다.

<https://account.vlab.dustbox.kr/deploy/>

계정은 하루만 제공이 됩니다. 좀 더 사용을 원하시는 경우, 저에게 메일로 알려주시면 문의 주신 메일 주소로 한 달 사용이 가능하도록 계정 생성 후 전달 해드립니다.

별도로 사용하는 퍼블릭 클라우드는 없습니다. 랩의 모든 사양은 공개가 되어 있으니, 본인 노트북 혹은 퍼블릭 클라우드에서 구성하여도 상관 없습니다.

다만, 퍼블릭 클라우드 구성 시, 비용이 꽤나 발생한다는 부분 인지하시길 바랍니다.

# 강사소개

소개

# 강사

이름: 최 국현

소속: 프리랜서

메일주소: tang@dustbox.kr

홈 페이지: tang.dustbox.kr, 아직 구성 중 입니다.

리눅스 컨설팅이나 혹은 강의 문의가 있으시면 언제든지 연락 주세요.

# 고마운 회사

장소지원

# 장소 협찬

이 자리를 빌어서 **왕 희웅 부장님, 조 대근 대리님**에게 다시 한번 감사 드립니다.

1. 이 부트캠프는 솔 데스크 × 공유개방을 통해서 강의실 제공을 받고 있습니다.
2. 수세 코리아에서 작은 기념품을 제공해 주셨습니다.
3. (주)쿠도에서 예상치 못한 다과를 지원해 주셨습니다.

미리 앞서 영상으로 안내 드렸지만, 간단하게 명함과 이름 및 서명만 미리 부탁 드립니다. 이를 통해서 추후 무료 세미나 안내를 드릴 예정입니다.

# 잠깐 교육 홍보

저의 돈벌이 :)

# 교육 일정

7월 및 8월에 아래와 같은 일정으로 강의가 있습니다.

업체	일정	강의	연락처	가격
개인	7월14일 ~ 16일	테크톤 기본	tang@dustbox.kr	50 만원(할인)
개인(수세)	7월22일 ~ 25일	Rancher+Harvester	tang@dustbox.kr	120 만원(할인)

# 교육 일정

**페스트레인 및 수세는 각 업체에서 비용 처리하며, 개인은 제가 직접 영수증 발급 합니다.**

개인 과정은 가끔 오픈 합니다. 필요하신 교육 과정이 있으신 경우(인프라 및 개발 포함) 편하게 메일 주시면 가/불가 여부를 안내 드리도록 하겠습니다. 강의가 필요하신 경우 직접 혹은 저의 교육 파트너

분류	설명
개인	제가 직접 진행하는 과정입니다. 모든 과정은 <u>온라인으로</u> 진행이 됩니다.
페스트레인	저와 함께 진행하는 과정입니다. 이 과정은 <u>오프라인으로</u> 진행이 됩니다.
수세	수세 코리아와 함께 기획하여 진행하는 과정입니다. 일반적으로 <u>오프라인으로</u> 진행이 됩니다. 결재 업체가 따로 있기 때문에, 메일 주시면 따로 안내 드리도록 하겠습니다.

# 챕터0

일단, 앤서블 활용

# 시스템 엔지니어에게 개발 능력이 필요한 이유

많은 사람들은 흔히 시스템 엔지니어가 하드웨어 및 인프라 구성에만 집중하고 소프트웨어 개발과는 거리가 멀다고 생각한다. 하지만 현대의 IT 환경에서는 시스템 엔지니어 역시 소프트웨어 개발자 못지 않은 개발 능력을 갖추는 것이 점점 중요해지고 있다. 그 이유는 다음과 같다.

# 인프라 자동화와 코드형 인프라(IaC)의 확산

- 전통적인 인프라 관리 방식은 GUI 또는 CLI 명령으로 수작업을 수행했다면, 최근의 환경은 Ansible, OpenTofu, Pulumi, Crossplane과 같은 IaC 도구를 통해 인프라를 코드로 관리하고 있다.
- IaC는 사실상 코드 작성을 필요로 하기 때문에, 시스템 엔지니어도 코드 관리 능력, 버전 관리(Git 등), 코드 리뷰 등의 개발 프로세스를 잘 알아야 한다.
- 특히 복잡한 클라우드 환경과 Kubernetes 생태계에서는 IaC를 통한 자동화가 필수다.

# 트러블 슈팅과 모니터링 자동화

- 시스템 문제 발생 시 단순히 로그만 보는 게 아니라, 로그 분석이나 시스템 점검, 장애 원인 분석 등을 자동화하는 프로그램을 개발하는 것이 일반화되고 있다.
- 이를 위해 Go 언어나 Python을 활용해 로그 분석, 모니터링 도구를 개발하거나 API 기반의 시스템 자동화를 구현하는 능력이 필요하다.
- Go 언어는 안정적이고, 성능이 뛰어나며, 호환성 유지가 잘 되기 때문에 시스템 엔지니어에게 인기가 많다.

# 클라우드 네이티브 환경의 복잡성

- Kubernetes와 같은 클라우드 네이티브 환경은 단순한 운영만이 아닌 운영과 개발이 융합된 DevOps 환경을 요구한다.
- 시스템 엔지니어가 Kubernetes의 CRD(Custom Resource Definition)를 작성하거나, Helm Chart, Operator를 개발할 때, 필연적으로 프로그래밍 능력이 요구된다.
- 이러한 작업은 더 이상 인프라 관리가 아니라, 개발과 운영이 함께 이루어지는 영역이다.

# 시스템 최적화 및 성능 튜닝

- 성능 분석 도구나 벤치마크 도구를 직접 제작하거나 튜닝할 필요가 있다.
- 시스템 엔지니어가 스스로 성능 측정 프로그램이나 스크립트를 개발하고 운영하면, 보다 정확한 분석과 빠른 문제 해결이 가능해진다.

# 시스템 엔지니어 vs. 소프트웨어 엔지니어

구분	시스템 엔지니어	소프트웨어 엔지니어
주요 역할	인프라 설계, 운영, 유지보수 및 자동화	애플리케이션 개발, 알고리즘 설계 및 구현
기술 스택 예시	Linux, OpenStack, Kubernetes, Ansible, Terraform(OpenTofu), Go, Bash, Python	Java, Go, Python, JavaScript, DB, 알고리즘, 자료구조
핵심 역량	인프라 구축 및 운영, 자동화 및 스크립팅 능력, 네트워크, 보안, 장애 대응 능력	코드 품질 관리, 알고리즘 구현 능력, 비즈니스 로직 구현, 소프트웨어 설계 능력
필요한 개발 역량	IaC, 모니터링 자동화, API 기반 시스템 제어 및 연동을 위한 개발 역량	시스템 아키텍처 이해, 애플리케이션 성능 최적화, 비즈니스 로직 및 UI/UX 개발 능력

# Go 언어를 선택하는 이유

## 호환성 및 안정성

- Go 언어는 버전 간의 호환성을 엄격하게 유지하며, 장기적인 운영에 있어 문제가 적다.
- Python은 자주 업데이트되고 라이브러리 호환성 문제가 많아, 장기적인 인프라 운영 및 유보수에 어려움이 있다.

## 성능과 경량성

- Go는 성능이 뛰어나며, 메모리 사용이 효율적이고 빠르다.
- 컨테이너 및 클라우드 환경에 특히 적합하다.

## 컴파일 및 배포의 간편성

- Go는 단일 바이너리로 배포가 간단하고, 컨테이너 기반의 배포에 매우 적합하다.
- 복잡한 의존성을 관리하지 않아도 되며, 시스템 엔지니어가 운영 환경에서 매우 편리하게 관리할 수 있다.

# 정리 및 예제

강사의 설명에 따라서 Go언어와 Ansible를 어떻게 같이 활용하는지 확인하고, 계정을 직접 생성 및 구성한다.

IAC Lab 신청

이메일

비밀번호

**신청하기**

결과

# 챕터1

쿠버네티스

# 도구

대다수 바닐라 버전 쿠버네티스 설치는 보통 두 가지로 진행한다.

1. kubeadm with bash
2. kubeadm with ClusterConfiguration

혹은 이게 어려운 경우 다음과 같은 도구를 사용한다.

1. kubespray
2. kind
3. kops

# 문제점

하지만, 이러한 도구에 문제는 다음과 같은 문제가 있다.

1. 기본적인 클러스터만 구성
2. 사용자화가 어려운 부분
3. 회사 혹은 고객이 원하는 부분까지 상세하게 적용하기가 어려움

이러한 이유로, 바닐라 버전 쿠버네티스를 사용하는 경우 다음과 같은 도구를 사용한다.

# 일반적인 도구

일반적으로 많이 사용하는 IaC 도구는 다음과 같다.

1. Ansible
2. Puppet
3. OpenTofu(Terraform)
4. salt
5. shell script

현재 커뮤니티는 대다수가 Ansible 기반으로 자동화 도구를 많이 사용하고 있다. 그 이유는 Ansible core는 GPL 3 라이선스를 사용하기 때문에, 특정 밴더에 Lock-In 될 가능성이 매우 낮다.

Terraform도 IBM으로 인수가 되면서, 라이선스를 GPL로 전환 OpenTofu라는 이름으로 다시 릴리즈 하였다.

# 안좋은 소식

불행하게도(?) 커뮤니티에서 많이 사용하는 도구 ansible, terraform이 IBM에 인수가 되면서, 커뮤니티는 다른 자동화 도구로 Left Shift하는 경향이 보이고 있다.

현재는 대다수 개발자 그리고 인프라 엔지니어는 다음과 같은 도구로 마이그레이션을 하고 있다.

1. Pulumi
2. Crossplane
3. OpenTofu

# Pulumi

## 코드 기반 IaC 전환 흐름

- 2025년 Pulumi 블로그에 따르면 "developer-centric team"이 주로 선택하는 도구로 부상 중이며 HCL 기반 Terraform의 한계를 넘어선 생산성을 제공한다고 분석함
- 여러 미디어에서 "다양한 언어 지원 및 프로그래밍 모델 덕분에 복잡한 로직 구현 시 유지보수가 쉽다"고 평가됨 .

# Crossplane

쿠버네티스 네이티브 인프라 관리

- Spacelift 블로그에 따르면 "크로스플레인은 쿠버네티스 기반 워크플로에 자연스럽게 병합되는 툴"로 특히 네트워크, 스토리지 등 클라우드 리소스 조정에 적합하다는 평
- "GitOps 흐름 안에서 지속적 재조정(reconciliation)이 가능하다"는 점이 크게 주목받고 있음

# Ansible

Ansible은 여전히 구성·네트워크 장비 자동화에 널리 쓰임

- Reddit 네트워크 엔지니어 토론에서는 Ansible이 "SSH를 통해 VLAN, SNMP 같은 네트워크 설정을 직접 조작할 수 있는 유연한 인프라 자동화 도구"라 평가됨.
- 학계 연구에서는 Ansible 사용자들이 "성능 문제, 선언문 구조 미비, 디버깅 어려움"을 지적한 바 있음 .

# 결론

Pulumi와 Crossplane은 인프라/네트워크 엔지니어 사이에서 점점 더 선호.

Ansible은 구성·네트워크 관리 분야에서 여전히 강력한 도구지만, 성장 동력은 둔화.

결론적으로, 쿠버네티스 네트워크 엔지니어들도 "Pulumi와 Crossplane"을 적극 고려. Ansible은 특정 구성 작업에 집중해 사용하지만 전체 파이프라인에서는 자동화 도구의 일부분으로 사용.

# 랩

하지만, 아직 많이 사용하고 익숙한 Ansible 기반으로 진행 합니다.

```
# ansible-playbook -i inventory/hosts.ini site.yml
# kubectl get pod -n tekton-pipelines
# kubectl get pod -n kube-system
# kubectl cluster-info dump | grep -E -i
' "(pod|svc)CIDR": [[:space:]]*[\""]+', '
```

## 챕터2

CI도구 설치 자동화

# 쿠버네티스에서의 CI/CD

쿠버네티스 환경에서 CI/CD는 애플리케이션을 컨테이너 이미지로 빌드하고, 자동으로 테스트하며, 컨테이너 오케스트레이션 플랫폼에 배포하는 과정을 지속적으로 반복하여 자동화하는 방법이다.

쿠버네티스의 선언적이고 자동화된 운영 환경을 통해, 애플리케이션 배포 및 업데이트 과정을 간소화하고 장애 발생 시 빠르게 복구할 수 있다.

# 애자일에서의 CI/CD

애자일 방법론에서 CI/CD는 짧은 개발 주기(iteration)를 통해 지속적으로 소프트웨어 품질을 향상시키고 빠른 피드백을 확보하는 핵심 요소이다.

개발자들이 작성한 코드를 자주 통합(Integration)하고, 통합된 코드를 지속적으로 테스트하며, 준비된 결과물을 자동화된 프로세스를 통해 빠르고 안전하게 배포(Delivery/Deployment)한다.

이를 통해 애자일의 핵심 가치인 신속한 변화 대응, 고객과의 긴밀한 협력, 빠른 피드백 적용을 효과적으로 실현할 수 있다.

# 단계 개념

CI/CD는 다음과 같은 단계 개념을 제공한다.

단계	설명	특징
지속적 통합	개발자가 작성한 코드를 자주 중앙 저장소에 병합하여 즉시 자동 빌드 및 테스트	빌드 실패 조기 발견, 품질 유지
지속적 제공	테스트를 통과한 결과물을 언제든 배포 가능한 상태로 준비하고 유지	수동 승인 후 배포 가능 상태 유지
지속적 배포	배포 가능한 결과물을 추가 승인 과정 없이 실시간으로 자동 배포	빠른 서비스 피드백, 즉시 개선 가능

# 지속적 통합(CI, Continuous Integration)

개발자가 코드 변경 사항을 중앙 저장소에 병합할 때마다 자동으로 빌드와 단위 테스트를 실행한다.

코드 품질을 빠르게 검증하여 머지(Merge)/충돌과 회귀 오류를 사전에 방지한다.

# 지속적 제공(CD, Continuous Delivery)

CI를 통과한 빌드 산출물을 스테이징(staging) 환경 혹은 프로덕션 환경에 배포 가능한 상태로 항상 유지한다.

수동 승인 또는 별도 절차를 통해 언제든 프로덕션에 릴리스 할 준비를 완료한다.

# 지속적 제공(CD, Continuous Delivery)

지속적 제공 단계에서 추가 승인 없이 자동으로 프로덕션 환경에 배포한다.

실서비스에 대한 즉각적인 피드백을 확보하여 빠른 개선 사이클을 실현한다.

# 대표 도구 정리

## 빌드 및 테스트 자동화(CI)

- Jenkins
- GitLab CI/CD
- GitHub Actions
- CircleCI

## 배포 파이프라인(CD)

- Argo CD
- Spinnaker
- GitLab CD
- Azure DevOps Pipelines

# 대표 도구 정리

## 인프라 자동화 및 구성 관리

- Terraform
- Ansible
- Helm (Kubernetes 패키징)

## 모니터링 및 피드백

- Prometheus
- Grafana
- ELK 스택

# 랩

아래처럼 플레이북을 실행한다.

```
# ansible-playbook -i inventory/hosts.ini playbook.yml
# podman container ls
# curl https://192.168.10.10:5000/v2/_catalog
# firefox https://192.168.10.10:3000
■ user: gogs
■ pass: gogs
■ URL: 192.168.10.10
# podman container rm --force --all
```

## Install Steps For First-time Run

If you're running Gogs inside Docker, please read [Guidelines](#) carefully before you change anything in this page!

### Database Settings

Gogs requires MySQL, PostgreSQL, SQLite3 or TiDB (via MySQL protocol).

**Database Type \*** PostgreSQL

**Host \*** 127.0.0.1:5432

**User \*** gogs

**Password \***

**Database Name \*** gogs

Please use INNODB engine with utf8\_general\_ci charset for MySQL.

**Schema \*** public

**SSL Mode \*** Disable

```
[root@iac-vm1 ci-tools-podman]# podman container ls
CONTAINER ID   IMAGE                               COMMAND             CREATED            STATUS              PORTS
          NAMES
17f1c67f0f0c  docker.io/library/registry:2    /etc/docker/regis...  4 minutes ago   Up 4 minutes   0.0.0.0:5000->5000/tcp
               registry
ca4e4e58c8de  docker.io/gogs/gogs:latest    /usr/bin/s6-svsc...  4 minutes ago   Up 4 minutes   0.0.0.0:3000->3000/tcp, 0.0.0.0
               10022->22/tcp  gogs
[root@iac-vm1 ci-tools-podman]# curl https://192.168.10.10:5000/v2/_catalog
{"repositories":[]}
```

# 마지막장

애플리케이션 빌드 및 배포

# TEKTON

테크톤은 CNCF에서 공식으로 지정한 Continuous Delivery 도구이다.

테크톤은 엄밀히 말하자면 절차적인 자동화 도구라고 보시면 된다. 테크톤은 Continuous Integration 기능도 제공하나, SCM에 구성이 되어 있는 소스코드를 가져와서 이미지 빌드를 할 수 있도록 지원한다.

테크톤은 개발자에게 다음과 같은 기능 제공한다.

1. build
2. test
3. deploy

테크톤에서 모든 자원은 쿠버네티스와 동일하게 YAML 기반으로 제공한다. 코드 기반으로 각각 작업들을 구성하며, 해당작업을 작업공간 및 파이프라인을 통해서 처리가 가능하다. 개발자가 손쉽게 YAML 코드 기반으로 애플리케이션을 서비스로 배포가 가능하다.

# TEKTON

테크톤은 CNCF사이트 소개가 되어 있다. **NONE-PROFIT 오픈소스 프로젝트**이다. 테크톤은 CNCF에서 아직 GRADUATED는 아니지만, CNCF에서 오픈소스로 보증하는 **프로젝트**이다.

**ArgoCD**는 CNCF에서 **GRADUATED**된 프로젝트이다.

The image shows two screenshots from the Cloud Native Computing Foundation (CNCF) website. The left screenshot is for the Continuous Delivery Foundation (CDF), which is NOT PROFIT. It shows a team size of 1 - 10 employees. The right screenshot is for the Argo project, which is GRADUATED. Argo is described as a Kubernetes-native tool for running workflows, managing clusters, and doing GitOps right. It has a maturity timeline from SANDBOX to INCUBATING to GRADUATED, with a specific date of 2020-03-26. Both screenshots include links to GitHub repositories and Apache License 2.0 information.

# TEKTON VS JENKINS

테크톤과 젠킨스는 기능 및 역할이 다르다. 젠킨스는 **플러그인 및 혹은 로컬 언어(Local Language)**를 사용해서 작업 순서를 구성한다.

젠킨스 **Groovy**라는 스크립트 명령어를 통해서 파이프라인 및 작업 수행이 가능하지만, 간단하게 파이프라인 및 자바 JDK와 같은 도구를 지원한다. 다른 의미로 테크톤 보다 더 제한적인 범위로 지원한다.

처음 시작하는 사용자에게는 학습 난이도가 낮은 테크톤 기반으로 구성을 권장하며, 추가적인 기능이 필요한 경우 플러그인 제작 및 설치가 별도로 필요하지 않는다. 또한, YAML문법을 사용하기 때문에 젠킨스의 Groovy언어 문법 및 구조체를 복잡하게 학습할 필요가 없다.

# TEKTON VS JENKINS

jenkins Groovy 문법 형식은 아래와 같다. 하양식이 아닌, 선언 및 문법을 통해서 Groovy코드를 작성해야 한다. 물론, 선언 형식에 대해서도 학습이 필요하다. 그와 반대로 테크톤은 YAML형식으로 단순하게 선언 및 작성이 가능하다.

```
node {  
    git url: 'https://github.com/jfrogdev/project-examples.git'  
    def server = Artifactory.server "SERVER_ID"  
    def downloadSpec =  
        '''{  
            "files": [  
                {  
                    "pattern": "libs-snapshot-local/*.zip",  
                    "target": "dependencies/",  
                    "props": "p1=v1;p2=v2"  
                }  
            ]  
        }'''
```

# TEKTON VS JENKINS

기능 비교하면 다음과 같다.

항목	Jenkins	Tekton
개발 주체	Jenkins Community (원래는 Kohsuke Kawaguchi)	Google + CD Foundation
아키텍처	모놀리식(Monolithic), 플러그인 기반	쿠버네이티브(Kubernetes Native), CRD 기반
설치/운영 방식	독립 실행형 서버 또는 컨테이너	Kubernetes 클러스터 내에서 실행되는 파이프라인 리소스
파이프라인 정의	Groovy 기반 DSL (Jenkinsfile)	YAML 기반 CRD (Pipeline, Task 등)
확장성	플러그인을 통한 확장 가능, 그러나 종속성 충돌 위험	Kubernetes 자원을 활용한 확장 (Pod, PVC 등)
CI/CD 목적성	CI 중심에서 출발 → CD는 추가 확장	CD를 염두에 둔 구조, 특히 GitOps 및 배포 중심

# TEKTON VS JENKINS

앞에 내용 계속 이어서…

항목	Jenkins	Tekton
보안	많은 플러그인은 권한 문제 발생 가능	Kubernetes의 RBAC 기반 보안 정책 활용 가능
사용성	GUI 친화적, 설정 직관적	CLI/yaml 중심으로 비교적 학습 곡선 있음
에이전트 실행	Jenkins 노드로 실행 (Master-Agent 구조)	각 Task가 Pod로 실행, 병렬 처리 용이
상태 저장	Jenkins 자체 DB 사용 (또는 외부 DB)	Kubernetes 리소스로 상태 관리 (ConfigMap, PVC 등)
커뮤니티 및 문서	매우 활발하고 오래된 커뮤니티	상대적으로 작지만 CNCF 지원 및 문서 지속 증가
적합한 환경	소규모~대규모 전통적인 CI/CD 환경	클라우드 네이티브, 쿠버네티스 중심의 CD 환경
예시 사용처	GitHub Actions과 연동한 기존 서버 중심 파이프라인	GKE, OpenShift 등 쿠버네티스 플랫폼 기반 자동화 파이프라인

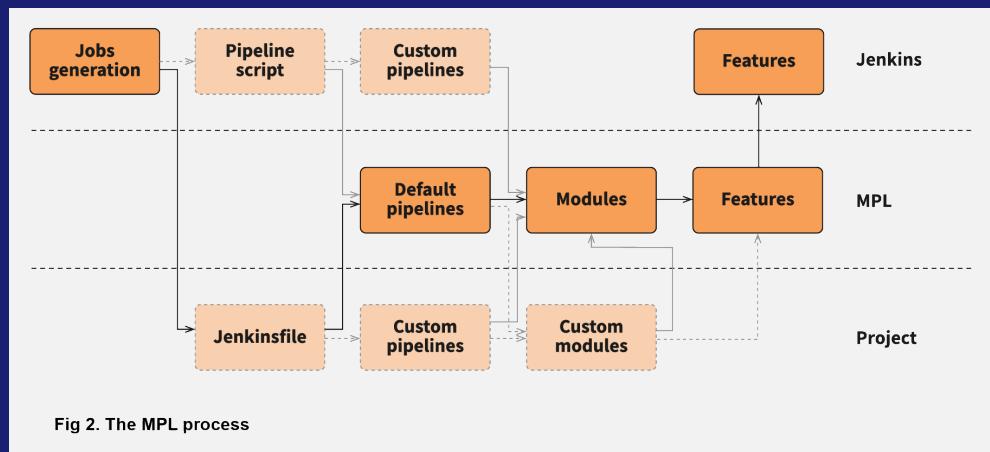
# TEKTON VS JENKINS

테크톤과 젠킨스의 제일 큰 차이점은 동작 방식이다.

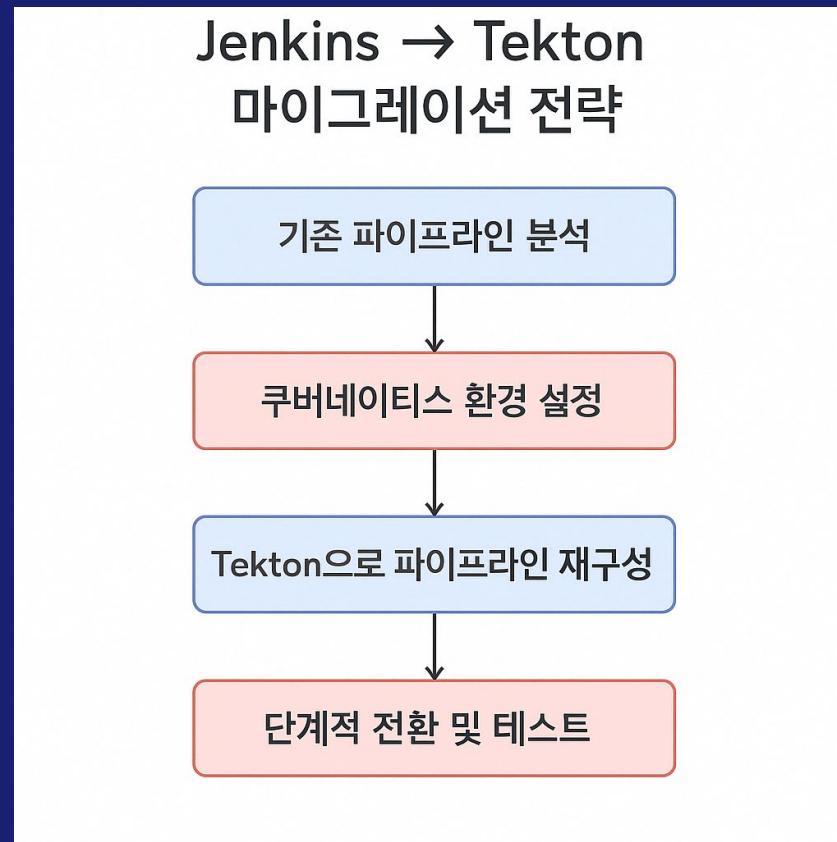
**젠킨스는** 쿠버네티스와 통합된 형태가 아니다. 본 기능은 자바 빌드용도 사용하다가 쿠버네티스와 통합된 상태이다. 그러한 이유로 젠킨스는 완벽하게 쿠버네티스와 통합이 되지 않는다.

**테크톤은** 기본적으로 쿠버네티스 클러스터 기반으로 구성 및 작성이 되었다. 젠킨스보다 손쉽게 사용하며, 별도로 관리 모듈을 만들지 않고 사용이 가능하다.

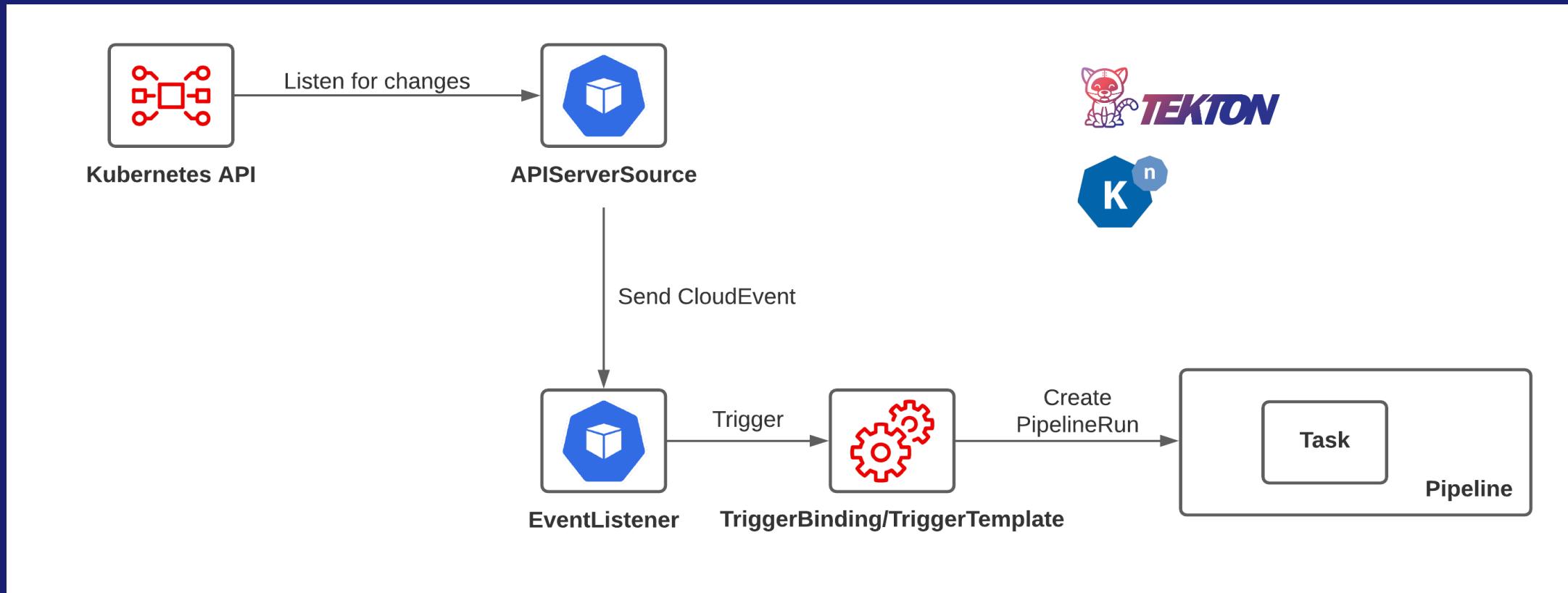
<https://www.jenkins.io/blog/2019/01/08/mpl-modular-pipeline-library/>



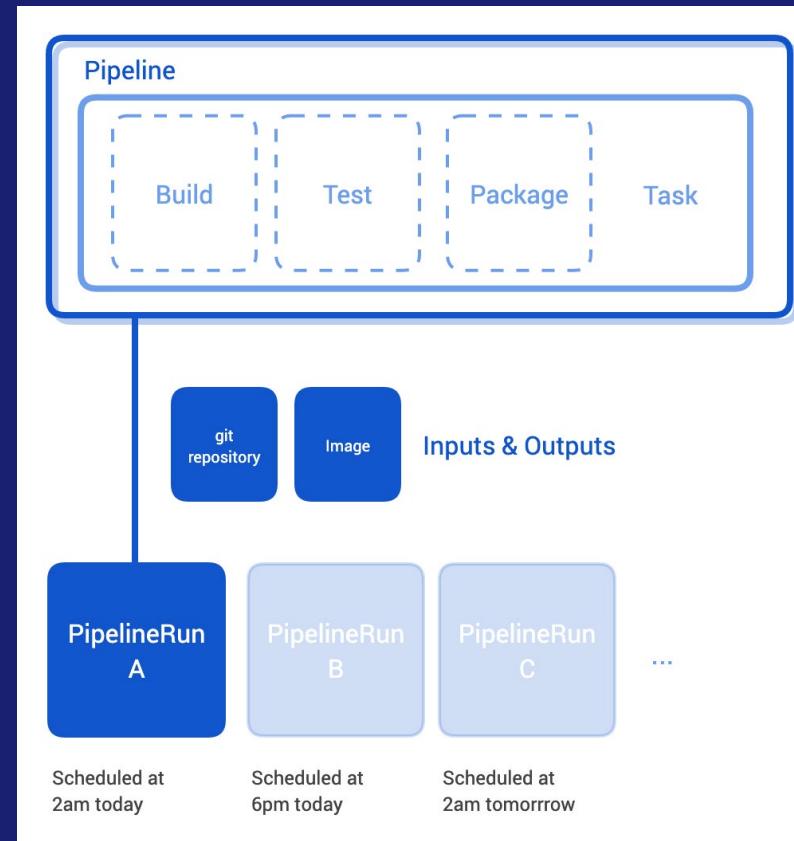
# マイグレーション



# IN KUBERNETES



# TEKTON PIPELINE



# 클라우드 네이티브에서 CI/CD

CI/CD

Jenkins/Tekton

2025-07-05

# 클라우드 네이티브

클라우드 네이티브는 수많은 컴퓨팅 개념 중 하나이다. 클라우드 네이티브에 제일 큰 차이점은 대다수 오픈소스 기반으로 개념들이 구성이 되어 있으며,

이를 기반으로 Cloud Native Computing Foundation(CNCF)라는 파운데이션을 만들었다. 테크톤은 CD영역에서 포함이 되며, 이를 통해서 K-Native기반의 자동화를 구성한다.



# CNCF/K-Native

많이 혼돈 Cloud Native와 K-Native이다. K-Native는 쿠버네티스 기반으로 서비스 구성이 목적이다.

K-Native는 컨테이너 기반으로 다음과 목적을 가지고 있다. 클라우드 네이티브는 국가 혹은 기업마다 목표 및 목적 이 다르다.

- **Simpler Abstraction:** YAML기반으로 CRD구성이 가능하다.
- **Auto Scaling:** 오토 스케일링 기반으로 0부터 확장 혹은 축소가 가능하다.
- **Progressive Rollouts:** 롤 아웃 상태를 전략에 따라서 구성이 가능하다.
- **Event Integration:** 모든 소스의 이벤트에 대해서 핸들링이 가능하다.
- **Handle Events:** 작업 이벤트를 트리거를 통해서 관리한다.
- **Pluggable:** 쿠버네티스에 확장 기능 추가가 가능하다.

# CR/CRD

## CR

보통 Custom Resource라고 부르며, 쿠버네티스 API를 통해서 생성이 된다.

대표적인 자원은 Pod, ConfigMap 그리고 Secrets이 있다. CR자원은 별도로 설치 및 구성이 필요 없으며 쿠버네티스 클러스터에서 지원 및 제공한다.

## CRD

CRD Custom Resource Definition의 약자이다. 쿠버네티스 네이티브 API가 아닌, 추가적으로 설치한 애플리케이션에 자원을 생성 시, CRD를 통해서 구성한다. 테크톤은 CRD를 통해서 자원을 생성 및 구성한다.

# STEP

기본자원

2025-07-05

# STEP

**단계(Step)**은 제일 기본적인 유닛이며, 이를 기반으로 파이프라인 구성이 된다. 최소 한 개의 작업이라도 단계에 구성이 되어야 한다. 이를 통해서 CI/CD 작업 수행 단계를 효율적으로 구성 및 운영이 가능하다.

각 단계에서는 명령어를 통해서 작업이 수행이 되는데, 예를 들어서 이미지 빌드를 위해서 소스코드를 내려받기 후 컴파일 과정이 필요하면, 이 부분을 단계에 명시한다. 각 단계에는 일반적으로 명령어를 통해서 어떠한 작업을 수행 할지 명시한다. 자세한 내용은 뒤에서 더 다룬다.

```
spec:
```

```
  steps:
```

```
    - image: quay.io/centos/centos
```

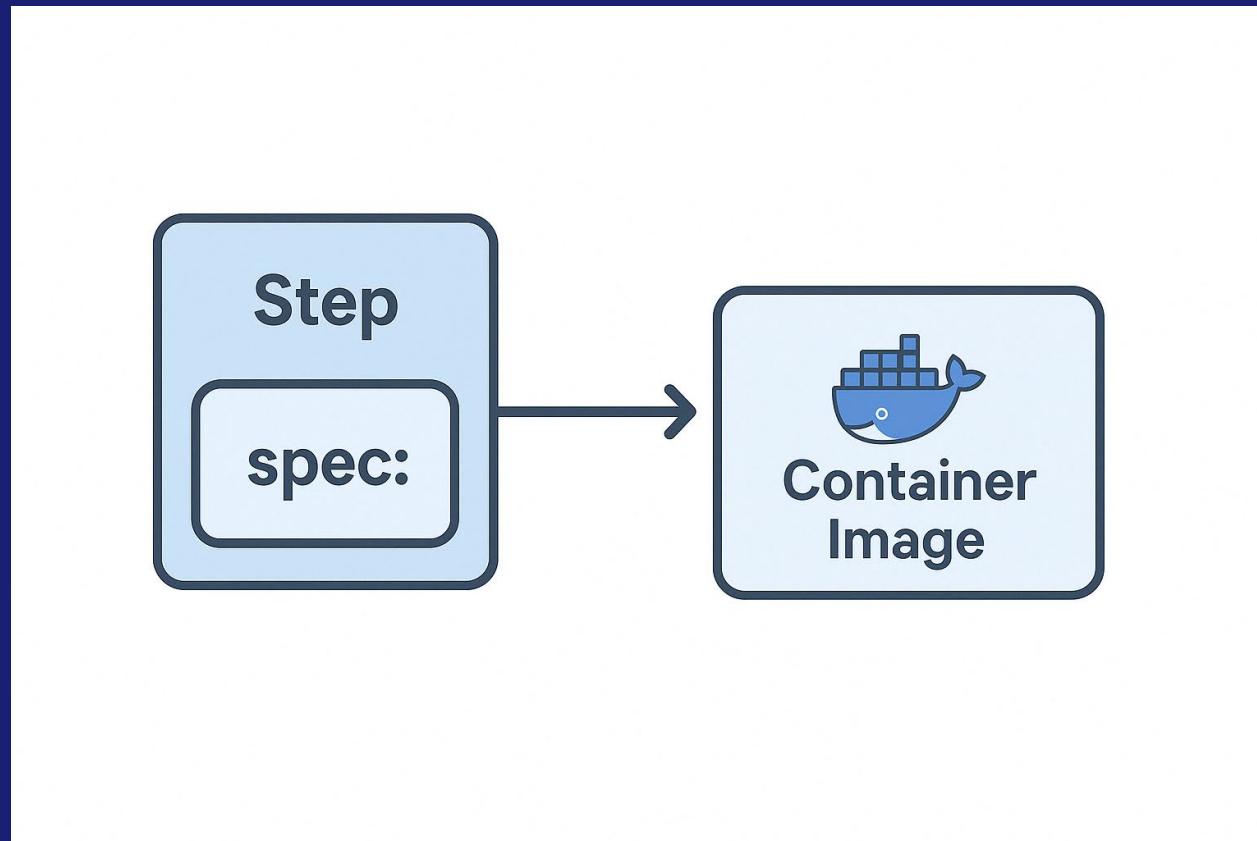
```
      command:
```

```
        - /bin/bash
```

```
        - -c
```

```
        - echo "Hello World"
```

# STEP



# TASK, PIPELINE

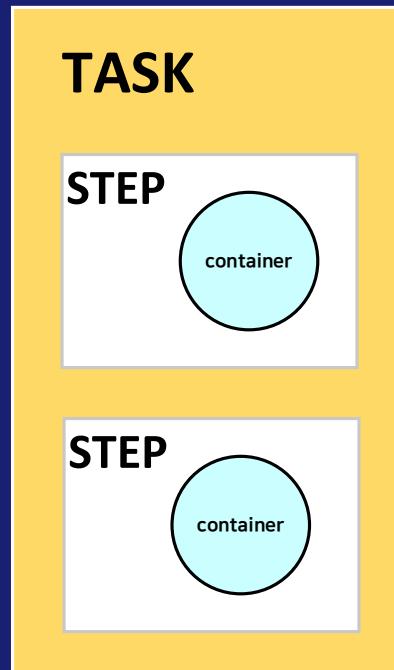
기본 자원

2025-07-05

# TASK

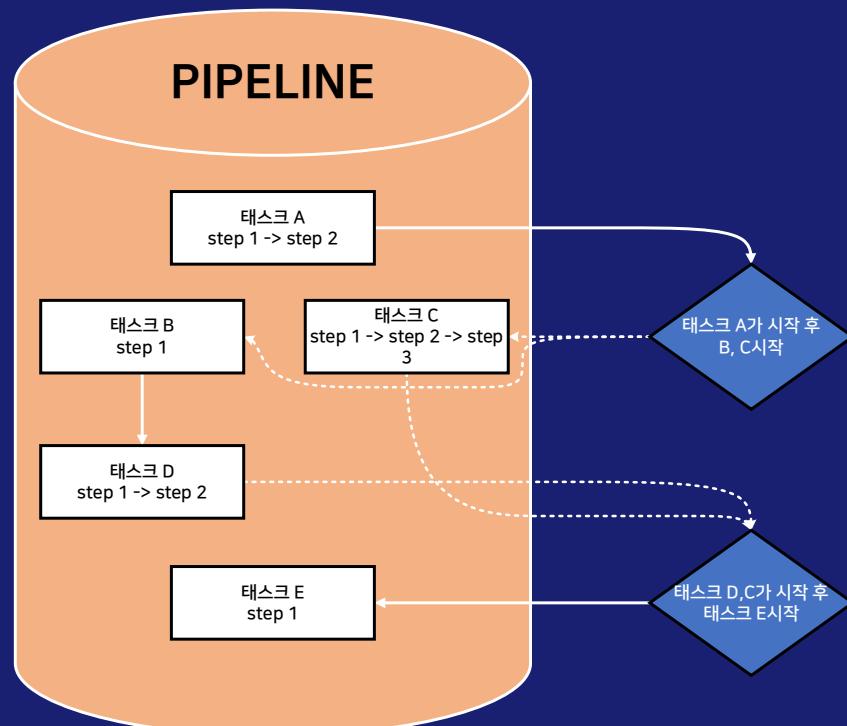
파이프 라인은 작업의 순서를 정하는 자원이다.

쉽게 생각하면 GW-BASIC처럼, 각각 작업에 순서를 걸어서 순차적으로 작업을 수행한다. 파이프 라인은 추상적인 자원이며, 이 자원은 기본적으로 태스크를 감싸고 있다.



# PIPELINE

모든 **스텝** 및 **태스크**에는 작업 순서가 있기 때문에, 작성된 순서대로 작업이 하나씩 수행이 된다. 이러한 형태를 가지고 있는 자동화 도구는 **앤서블/솔트**와 같은 도구들이 있으며, 이들은 테크톤과 동일하게 **YAML**기반으로 작업을 작성 및 구성하게 된다.



# PIPELINE

아래는 파이프라인 생성 YAML코드이다. 파이프라인은 반드시 어떤 작업(task)를 수행할지 명시해야 한다.

```
# vim first-pipeline.yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: first-pipeline
spec:
  tasks:
    - name: first
      taskRef:
        name: first-task
```

# TASK

태스크(tasks)는 파이프라인에 안에서는 작업에 대한 격리를 한다.

여기서 말하는 작업은 단계(step)를 이야기 하며, 각각 단계들은 태스크에서 수행이 된다. 각 태스크는 단계들의 순서(sequence)를 가지고 있으며, 이를 통해서 하나의 작업을 통해서 모든 작업을 수행한다.

쿠버네티스 클러스터에서 Pod를 생성하기 위해서 이미지 작업 및 Pod생성과 같은 과정이 필요하다. 이러한 부분을 각각 단계로 구성하여 작업을 수행 후, 최종적으로 API를 통해서 Pod생성 및 PV/PVC와 같은 자원을 구성 및 연결을 수행한다.

이러한 작업 단계를 모아서 동작하는 영역이 파이프라인이며, 작업(task)는 단계(step)에 명시된 작업들을 분리 및 격리하여 수행 할 수 있도록 한다. 작업을 구성하면, 이들은 파이프 라인을 통해서 구성 및 실행이 된다.

# TASK YAML

아래는 간단하게 태스크(task)를 생성하는 YAML이다.

```
# vi first-task.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: first-task
spec:
  steps:
  - name: first-task
    image: quay.io/centos/centos
    command:
      - echo "hello world"
```

# 생성 자원 확인

생성된 자원은 아래와 같이 명령어로 확인이 가능하다.

```
# tkn pipeline list  
# tkn task list  
# tkn pipeline start first-pipeline  
# tkn pipeline list
```

NAME	AGE	LAST RUN	STARTED	DURATION	STATUS
first-pipeline	14 minutes ago	first-pipeline-run-86x7v	19 seconds ago	---	Running

# 기본 자원

Stepping

Task

Pipe

2025-07-05

# STEP

단계(스텝)은 특정 작업을 수행하는 부분이다. 단계는 다음과 같이 작업을 수행한다.

1. 소스코드 내려받기
2. 컴파일 및 명령어 실행
3. 프로그램 패키징

총 3단계를 통해서 작업을 한다. 이미지가 비공개 저장소에 있는 경우, **ImagePullSecret**를 통해서 내려받기가 가능하다. 작성 방법은 아래와 같이 작성이 가능하다.

# STEP

아래와 같이 작업을 생성한다.

```
# vi demo-task-1.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-task-1
spec:
  steps:
    - image: quay.io/centos/centos:stream9
      command:
        - /bin/bash
        - -c
        - echo "Hello World"
```

# STEP

생성한 **demo-task-1**를 아래와 같이 생성한다.

```
# kubectl apply -f demo-task-1.yaml
# tkn task ls
NAME          DESCRIPTION      AGE
demo-task-1
hello
# tkn task start demo-task-1
# tkn task start demo-task-1 --showlog
```

# STEP FOR MULTI STEP

한 개 이상의 단계가 있는 경우, 아래와 같이 작성이 가능하다.

```
# vi demo-multi-step.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-multi-step
spec:
  steps:
    - name: first
      image: quay.io/centos/centos:stream9
      command:
        - /bin/bash
        - -c
        - echo "First step"
```

# STEP FOR MULTI STEP

```
- name: second
  image: quay.io/centos/centos:stream9
  command:
    - /bin/bash
    - -c
    - echo "Second step"
```

```
# tkn task list
# tkn task start demo-multi-step
# tkn taskrun logs demo-multi-step-run-hlvgx -f
```

# STEP IMAGE

단계 구성 시, 사용하는 컨테이너 이미지는 사용자가 원하는 이미지로 변경이 가능하다. 여기서는 일반적으로 많이 사용하는 centos 이미지 기반으로 구성하였다. 이미지는 최소 한 개가 구성 및 선언이 되어야 한다.

문법은 아래와 같이 선언이 된다.

현재 연습 예제에서는 centos 기반으로 테스트를 수행 및 진행한다. 만약, 다른 배포판 사용을 원하는 경우, 다른 컨테이너 이미지를 사용하여도 된다.

- ```
- name: second  
image: quay.io/centos/centos
```

# STEP 실행 및 상태 확인

올바르게 실행이 되었는지 아래 명령어로 확인한다.

```
# tkn task list  
# tkn task start demo-multi-step --showlog  
# tkn task start demo-multi-step --showlog --no-color  
# kubectl get tasks  
# kubectl get taskruns
```

# STEP 스크립트 사용

다중 명령어를 실행하거나 혹은 스크립트를 실행하는 경우, 아래와 같이 처리가 가능하다.

```
- name: step-in-script
  image: quay.io/centos/centos:stream9
  script: |
    #!/usr/bin/env bash
    echo "Install a package"
    dnf install httpd -y
    dnf clean all
    echo "All commands ran!"
```

# STEP IN SCRIPT

위의 내용을 코드로 변경하면 아래와 같다.

```
$ vi demo-step-script.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-step-script
spec:
```

# STEP IN SCRIPT

위의 내용 계속 이어서…

```
steps:  
- image: quay.io/centos/centos:stream9  
script: |  
#!/usr/bin/env bash  
echo "Install a package"  
dnf install httpd -y  
dnf clean all  
echo "All commands ran!"
```

# STEP IN SCRIPT

아래 명령어 적용 및 확인한다.

```
# kubectl apply -f demo-step-script.yaml  
# tkn task list  
# tkn task start demo-step-script --showlog
```

# TASK PARAMETER

앞에서 간단하게 단계와 작업을 동시에 작성 및 실행 하였다. 하지만, 매번 작업을 구성할 때마다 단계에 들어가는 값을 변경할 수 없기 때문에, 재사용을 위해서 파라메터 형식으로 변경한다.

파라메터, 즉 변수 형태로 하는 경우, "param" 항목에서만 변경하면서 사용이 가능하다. 좀 더 효율적으로 운영 및 구성이 가능하다.

위와 같이 값의 이름, 그리고 유형을 적어주면, 쉘 텍스트 입력 받는 형식과 비슷하게 값을 읽어온다. 예로, 앞에서 사용하였던 명령어 실행을 예로 든다.

## params:

- **name: username**  
**type: string**

# TASK PARAMETER

아래와 같이 작성하면, 변수를 통해서 좀 더 효율적으로 코드 사용이 가능하다.

**command:**

- /bin/bash
- -c
- echo "Hello \${params.who}"

# TASK PARAMETER

위의 두 개 예제를 가지고 다음과 같이 테크톤 작업형식으로 작성이 가능하다. 아래 슬라이드처럼 파일을 작성한다.

```
# vi demo-task-param.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-task-param
spec:
  params:
    - name: username
      type: string
```

# TASK PARAMETER

위의 내용 계속 이어서…

**steps:**

- image: quay.io/centos/centos:stream9

**command:**

- /bin/bash
- -c
- echo "Hello \$(params.username)"

# TASK PARAMETER

적용 후, 실행하면 다음과 같이 실행 및 확인이 된다.

```
# kubectl -f demo-task-param.yaml
# tkn task list
# tkn task start demo-task-param --showlog
? Value for param `username` of type `string`? tang
TaskRun started: demo-task-param-run-9rpcz
Waiting for logs to be available ...
[unnamed-0] Hello tang
# tkn task start demo-task-param --showlog -p username=tang
```

# TASK ARRAY PARAMETER

파라메터를 배열로 처리가 가능하다. 다만, 좀 더 복잡하게 코드를 구성해야 한다. 기존 코드에 아래와 같이 파일 이름을 변경 후 작성한다.

```
# vi demo-task-param-array.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-task-param-array
spec:
  params:
    - name: users
      type: array
```

# TASK ARRAY PARAMETER

파라메터를 배열로 처리가 가능하다. 다만, 좀 더 복잡하게 코드를 구성해야 한다. 기존 코드에 아래와 같이 파일 이름을 변경 후 작성한다.

## steps:

- name: list-users
  - image: quay.io/centos/centos:stream9
  - args:
    - \$(params.users[\*])
  - command:
    - /bin/bash
    - -c
    - for ((i=1;i<=\$#;i++)); do echo "\$#" "\$i" "\${!i}"; done

# TASK PARAMETER ARRAY

테크톤에 작업 등록 후 작업을 수행한다.

```
# kubectl apply -f demo-task-param-array.yaml
# tkn task list
# tkn task start demo-task-param-array --showlog --use-param-defaults
test1, test2, test3
```

# TASK DEFAULT PARAMETER

파라미터에 기본값 설정이 필요한 경우 `default`라는 변수 키워드를 사용한다. 미리 사용할 기본값을 미리 작성한

```
# vi demo-task-param-default.yaml
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: demo-task-param-default
spec:
  params:
    - name: users
      type: array
  default:
    - test1
    - test2
    - test3
```

# TASK DEFAULT PARAMETER

앞에 내용 계속 이어서…

```
steps:  
- name: list-users  
  image: quay.io/centos/centos:stream9  
  args:  
  - ${params.users[*]}  
  command:  
  - /bin/bash  
  - -c  
  - for ((i=0;i<=${#};i++)); do echo "$#" "$i" "${!i}"; done
```

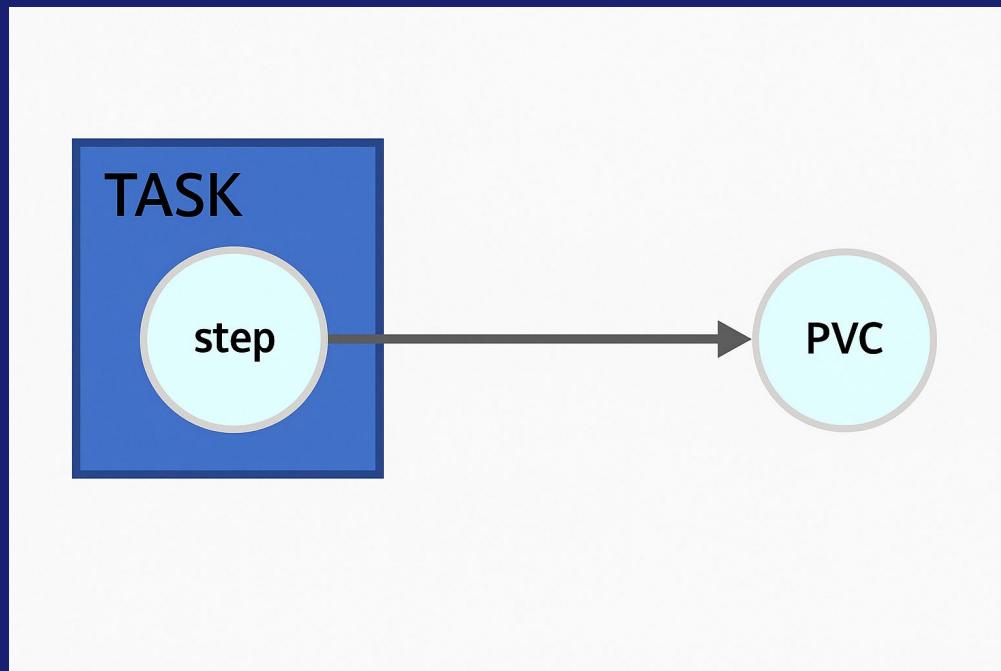
# TASK DEFAULT PARAMETER

기본 변수 값을 쿠버네티스에 등록 후 실행한다.

```
# kubectl apply -f demo-task-param-default.yaml  
# tkn task list  
# tkn task start demo-task-param-default --use-param-defaults --showlog
```

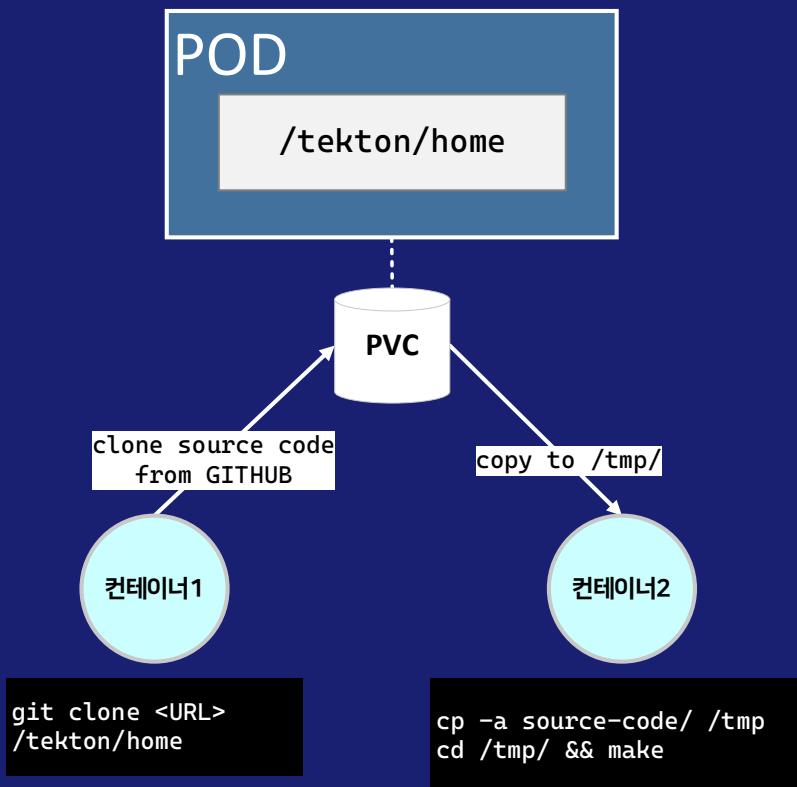
# 공유 데이터

테크톤이 동작하면 컨테이너가 하나의 **포드(Pod)**에서 동작하기 때문에, 데이터 공유가 가능하다. 데이터를 공유하기 위해서 쿠버네티스의 **PV/PVC**를 통해서 공유가 가능하다.



# 공유 홈

테크톤의 고유한 디렉터리를 사용하여 POD에서 컨테이너간 데이터 공유가 가능하다.



# 랩

다음처럼 CD 구성 및 적용한다.

```
# kubectl create namespace blog
# kubectl config set-context --current --namespace blog
# tkn hub install task kubernetes-actions
# tkn hub install task git-clone
# tkn hub install task buildah
# kubectl apply -f tkn-build-www.yaml
# kubectl apply -f tkn-build-db.yaml
# kubectl apply -f storageclass.yaml
# kubectl apply -f pvc-codelab.yaml
# kubectl apply -f maven-rocky-build.yaml
# tkn pipeline start tkn-build-db --showlog
# tkn pipeline start tkn-build-www --showlog
```

# 랩

아래와 같이 확인이 가능하다.

```
# tkn pipeline list
NAME          AGE        LAST RUN      STARTED      DURATION
tkn-build-db  4 hours ago  tkn-build-db  2 hours ago  1m1s
tkn-build-www 4 hours ago  tkn-build-www  2 hours ago  2m7s
# curl https://192.168.10.10:5000/v2/_catalog
```

# 마무리

결론

# 정리

다음과 순서로 부트캠프를 정리합니다.

1. 쿠버네티스 KiKi 프로젝트, 락 플레이스(장경빈, 안현)
2. 부트캠프 2차 강의(날짜 미정, 쿠버네티스 플랫폼 비교(Okd, Kubernetes, Rancher, KubeSphere))
3. 부트캠프 3차 강의(날짜 미정, 20대 이상의 서버 관리 및 운영 방법(Ansible, Salt, OpenTofu))