THE IaC

목적

부트캠프 목적

부트캠프 목적

현재 많은 오픈소스가 활성화가 되어 있고, 이를 기반으로 많은 회사들이 제품을 구성 및 판매를 하고 있습니다. 많은 기업들은 **인프라/개발**에서 점점 IaC를 어떻게 활용하는지 고민이 많습니다.

이 부트캠프는 아래와 같은 도구 및 기술을 이야기 합니다.

1. Ansible/Sal/Pulumi
2. 쿠버네티스
3. 로키 리눅스(9.x)
4. ~~오픈스택 2025.1(추후 다룰 예정)~~

부트캠프에 현재 프로비저닝 부분은 제외가 되어 있다.

랩 환경

랩은 오픈스택 기반으로 제공이 되며, 아래 주소에서 IaC를 활용하여 오픈스택 계정 생성 후, 진행 합니다.

이 부분에 대해서는 뒤에서 별도로 설명 드리도록 하겠습니다.

`https://account.vlab.dustbox.kr/deploy/`

강사소개

소개

강사

이름: 최 국현

소속: 프리랜서 강사 및 엔지니어

- 수세/레드햇 제품 컨설팅
- 오픈소스 프로젝트 및 설계 컨설팅
- 국가 오픈소스 프로젝트 참여

메일주소: tang@dustbox.kr

홈 페이지: tang.dustbox.kr

컨설팅이나 혹은 강의가 필요하시면 언제든지 연락주세요.

고마운 회사

장소지원

장소 협찬

이번 2차 IaC 부트캠프는 **㈜에티버스** 그리고, 에티버스 러닝에서 제공 및 지원해 주셨습니다.

다시 한번, **에티버스 황 승억 부장님**에게 감사 드립니다.

1. 아마도 기념품이 나갈 수 있습니다.
2. 식사 및 다과는 **㈜에티버스에서** 제공해 주셨습니다.

챕터0

일단, 앤서블 활용

IaC

왜 IaC?

IaC

IaC(Infrastructure as Code)는 인프라스트럭처를 코드로 정의하고 관리하는 방식이다.

이전까지 서버, 네트워크, 스토리지 등의 인프라는 GUI나 수동 명령어를 통해 직접 설정해야 했지만, IaC는 이러한 구성을 코드 파일로 명시함으로써 **자동화된 배포와 일관된 운영이 가능하도록 한다**.

IaC는 보통 `YAML, JSON, HCL(HashiCorp Configuration Language)` 등 선언형 언어 또는 Python, Go 같은 절차적 언어를 사용하여 작성되며, Git 같은 버전 관리 시스템과 결합해 DevOps와 GitOps의 **핵심 구성요소로 작동한다**.

IaC

대표적인 도구로는 Terraform, Pulumi, Ansible, AWS CloudFormation 등이 있으며, 이들은 각기 다른 방식으로 클라우드나 온프레미스 자원을 선언하고 프로비저닝할 수 있도록 해준다.

IaC를 통해 조직은 수작업을 줄이고 오류 가능성을 최소화할 수 있으며, 테스트 및 검증된 코드를 기반으로 **지속적이고 재현 가능한 인프라 환경을 구성할 수 있게 된다**.

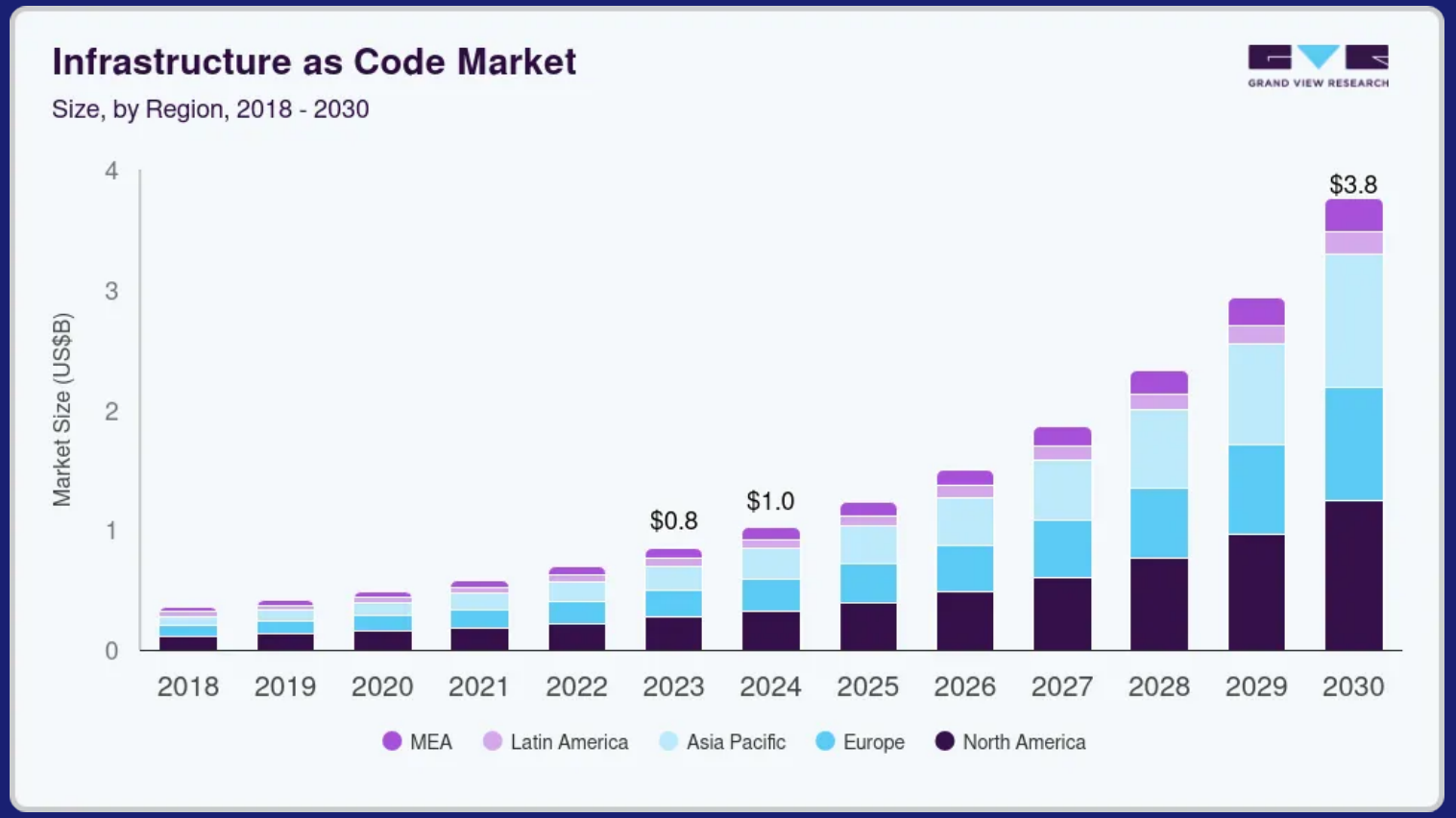
결론적으로, IaC는 **인프라를 코드처럼 다룰 수 있도록 하여 클라우드 시대의 인프라 운영을 근본적으로 변화시키는 패러다임이다**.

IaC 성장장

현재 IaC는 클라우드와 비슷하게 빠르게 성장하는 도구 시장 입니다.

소스	기간	시장 규모 범위	CAGR
Fortune Business Insights	2022→2030	0.76 → 3.30 B USD	20.3%
MarketsandMarkets	2022→2027	0.8 → 2.3 B USD	24.0%
Grand View Research	2023→2030	0.85 → 3.76 B USD	24.4%

GVR 자료


Infrastructure as Code Market
Size, by Region, 2018 - 2030
GRAND VIEW RESEARCH
Market Size (US$B)
4
3
2
1
0
$0.8
$1.0
$3.8
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
MEA
Latin America
Asia Pacific
Europe
North America

IDC/Gartner 의견

IDC 보고서

IaC가 클라우드(IaaS/PaaS) 거버넌스 및 개발자/운영팀의 셀프서비스 플랫폼으로 핵심 역할을 수행하고 있다고 강조.

Gartner Magic Quadrant 보고서

보고서에서는 (Cloudflare 사례 포함) IaC 지원 API 통합이 규제·보안·자동화를 위한 SASE/SSE 플랫폼의 주요 요건으로 언급됨.

인프라 자동화

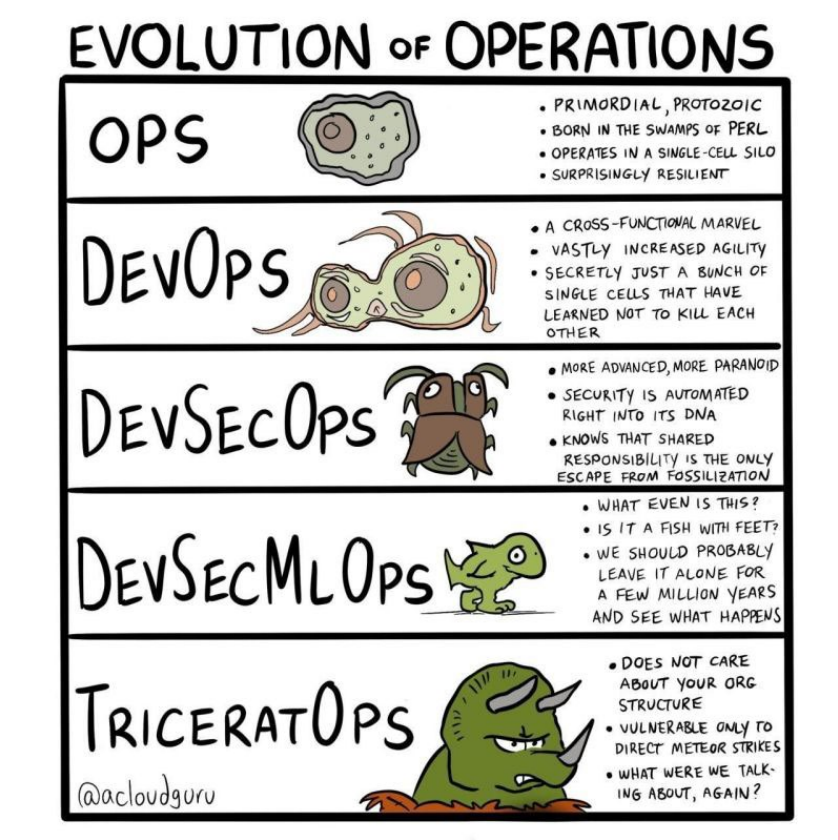
- 전통적인 인프라 관리 방식은 GUI 또는 CLI 명령으로 수작업을 수행했다면, 최근의 환경은 Ansible, OpenTofu, Pulumi, Crossplane과 같은 IaC 도구를 통해 인프라를 코드로 관리하고 있다.
- IaC는 사실상 코드 작성을 필요로 하기 때문에, 시스템 엔지니어도 코드 관리 능력, 버전 관리(Git 등), 코드 리뷰 등의 개발 프로세스를 잘 알아야 한다.
- 특히 복잡한 클라우드 환경과 Kubernetes 생태계에서는 IaC를 통한 자동화가 필수다.

DEVOPS 엔지니어?

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.
- GitOps는 DevOps 철학의 일부분이다. 더 이상 시스템 엔지니어가 다루는 인프라는 제품의 일부분에 포함이 된다.
- 퍼블릭/프라이빗 심지어 온프레미스 네트워크 및 인프라 장비에도 IaC는 이제는 표준으로 적용이 된다.

DEVOPS 엔지니어?


EVOLUTION OF OPERATIONS
OPS
• PRIMORDIAL, PROTOZOIC
• BORN IN THE SWAMPS OF PERL
• OPERATES IN A SINGLE-CELL SILO
• SURPRISINGLY RESILIENT
DEVOPS
• A CROSS-FUNCTIONAL MARVEL
• VASTLY INCREASED AGILITY
• SECRETLY JUST A BUNCH OF SINGLE CELLS THAT HAVE LEARNED NOT TO KILL EACH OTHER
DEVSECOPS
• MORE ADVANCED, MORE PARANOID
• SECURITY IS AUTOMATED RIGHT INTO ITS DNA
• KNOWS THAT SHARED RESPONSIBILITY IS THE ONLY ESCAPE FROM FOSSILIZATION
DEVSECMLOPS
• WHAT EVEN IS THIS?
• IS IT A FISH WITH FEET?
• WE SHOULD PROBABLY LEAVE IT ALONE FOR A FEW MILLION YEARS AND SEE WHAT HAPPENS
TRICERATOPS
• DOES NOT CARE ABOUT YOUR ORG STRUCTURE
• VULNERABLE ONLY TO DIRECT METEOR STRIKES
• WHAT WERE WE TALKING ABOUT, AGAIN?
@acloudguru

시스템 최적화

- 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.
- IaC는 시스템 최적화에 포함이 되어 있다. 이를 통해서 시스템 운영 환경을 최적화 한다.

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

마지막으로

현재 인프라 대다수 유명한 도구는 두 가지로 정리가 되어가고 있다.

1. Go
2. Rust

컨테이너 및 IaC 기술들은 현재 Go 언어로 정리가 되어가고 있다. 이러한 이유로, 시스템 엔지니어에게 Go 언어 학습을 매우 권장한다.

마지막으로

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 유지보수에 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.

마지막으로

대표 도구	언어
Docker	Go
Kubernetes	Go
Terraform	Go
Prometheus	Go
Helm	Go
CNI/CSI	Go

마지막으로

Go, Ansible로 만든 예제를 가지고 인프라 엔지니어 미래를 간단하게 살펴본다.

참고

https://www.fortunebusinessinsights.com/infrastructure-as-code-market-108777

https://www.marketsandmarkets.com/Market-Reports/infrastructure-as-code-market-115458264.html

https://www.grandviewresearch.com/industry-analysis/infrastructure-as-code-market-report

챕터1

쿠버네티스

도구

대다수 바닐라 버전 쿠버네티스 설치는 보통 두 가지로 진행한다.

1. kubeadm with bash
2. kubeadm with ClusterConfiguration

위의 도구가 어려운 경우, 쿠버네티스에서 제공하는 자동화 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
4. 자동화를 위한 자동화 코드 반영 및 수정이 어려움
5. 수정 후, 버전 유지보수에 대한 문제가 발생

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. Pulumi

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

Terraform도 IBM으로 인수가 되면서, 라이선스를 GPL로 전환 OpenTofu라는 이름으로 다시 **릴리즈(Fork)** 하였다.

안 좋은 소식

최근 Ansible과 Terraform이 IBM에 인수되면서, 오픈소스 커뮤니티 내에서는 벤더 종속성에 대한 우려가 확산.

이에 따라 커뮤니티는 자동화 도구 선택에 있어 'Left Shift' 흐름, 즉 보다 유연하고 클라우드 네이티브에 가까운 대안으로의 전환 경향을 보이고 있다.

현재 많은 개발자와 인프라 엔지니어들은 다음과 같은 도구로 마이그레이션을 고려하거나 이미 채택하고 있다.

- Pulumi – 프로그래밍 언어 기반의 선언형 IaC
- Crossplane – Kubernetes CRD 기반 인프라 구성
- OpenTofu – Terraform 오픈소스 포크, 벤더 독립성 강화

안 좋은 소식

하지만, 여전히 Ansible Core는 오픈소스에서 제일 많이 사용하는 IaC 도구. 또한, 커뮤니티에서 표준 IaC 도구로 선택을 많이 하고 있다.

툴	GitHub 활동성	StackShare 스택/팔로워	커뮤니티 성장성	커뮤니티 규모
Ansible	⭐⭐⭐⭐⭐	19.3K / 15.5K	매우 높음	최상위
Terraform	⭐⭐⭐⭐⭐	18.7K / 14.7K	매우 높음	Ansible과 격차 작음
SaltStack	⭐⭐⭐	—	보통	Ansible/Terraform보다 작음

Ansible

Ansible은 여전히 구성·네트워크 장비 자동화에 널리 쓰임

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

SaltStack

Salt는 고속 병렬 실행과 이벤트 기반 아키텍처 덕분에 대규모 시스템 관리에 강점을 지닌 자동화 도구로 사용됨

1. Reddit 및 시스템 운영 커뮤니티에서는 Salt가 "수천 대의 서버에 대해 병렬로 명령을 실행할 수 있는 유일한 오픈소스 툴"이라고 평가됨.
2. 커뮤니티 보고서에서는 Salt 사용자들이 "학습 곡선이 가파르고, 모듈 시스템이 복잡하며, 상태 정의의 직관성이 낮다"는 피드백을 공유함.

Pulumi

코드 기반 IaC 전환 흐름

1. 2025년 Pulumi 블로그에 따르면 "developer-centric team"이 주로 선택하는 도구로 부상 중이며 HCL 기반 Terraform의 한계를 넘어선 생산성을 제공한다고 분석함
2. 여러 미디어에서 "다양한 언어 지원 및 프로그래밍 모델 덕분에 복잡한 로직 구현 시 유지보수가 쉽다"고 평가됨 .

Crossplane

쿠버네티스 네이티브 인프라 관리

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

결론

1. Pulumi와 Crossplane은 클라우드 및 인프라 엔지니어 사이에서 점점 선호도가 올라가는 중.
2. Ansible은 네트워크 및 시스템 관리 분야에서 여전히 강력한 도구지만, **성장 동력은 둔화.**
3. Terraform은 라이선스로 인하여 Fork 이후, 점점 커뮤니티에서 거리를 두는 추세.

쿠버네티스 네트워크 엔지니어들도 **"Pulumi와 Crossplane"**을 적극 고려. Ansible은 특정 구성 작업에 집중해 사용하지만, 클라우드 파이프라인에서는 특정 영역에서만 사용하고, 전체적으로 적용이 안되는 중.

특히, 쿠버네티스를 지원하지만 앤서블 보다는 Tekton와 같은 CNCF IaC도구를 더 선호 함.

랩(Ansible)

강사와 함께 코드 리뷰하면서 랩을 진행 합니다. 설치는 앤서블 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:]]*"[^"]+",'
```

Salt/Pulumi

```
k8s_iac_example/
├── Pulumi.ts                    ← Pulumi로 kubeadm 기반 마스터/워커 구성
└── salt/
    ├── top.sls                  ← Salt 대상 분리 (마스터, 워커, 공통)
    └── kubernetes/
        ├── common.sls           ← kubeadm, kubectl 등 공통 패키지 설치
        ├── master.sls           ← kubeadm init 및 config 복사
        └── worker.sls           ← join 명령어를 통한 클러스터 참여
```

랩(Pulumi)

```
# curl -fsSL https://get.pulumi.com | sh
# export PATH=$PATH:$HOME/.pulumi/bin
# dnf install -y nodejs
# npm init -y
# npm install @pulumi/pulumi @pulumi/command
# pulumi login --local
# pulumi new typescript --force
# cp Pulumi.ts index.ts
# pulumi up
```

랩(Salt)

```
# dnf install -y salt-master salt-minion
# systemctl enable --now salt-master
# vi /etc/salt/minion
master: 192.168.1.10
id: k8s-master1
# vi /etc/salt/master
file_roots:
  base:
    - /root/codelab/bootcamp/IaC-bootcamp/k8s-Salt
# systemctl restart salt-master
```

랩(Salt)

```
# systemctl enable --now salt-minion
# vi /etc/salt/minion
master: 192.168.1.10
id: k8s-worker1
# systemctl restart salt-minion
# salt-key -L
# salt-key -A
# salt '*' state.apply
```

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.

애자일(DevOps)에서의 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.

인프라 엔지니어(시스템 엔지니어)의 역할은?

1. **플랫폼 구축 및 자동화**
   - 쿠버네티스, OpenStack, Harvester 같은 클라우드/가상화 플랫폼 구성
   - CI/CD 파이프라인과 IaC 도구(Terraform, Ansible, Pulumi 등) 자동화 구성
   - GitOps 도구(ArgoCD, Flux) 운영 및 최적화
2. **신뢰성 있는 인프라 운영**
   - SLA를 고려한 고가용성(HA) 구성
   - 모니터링/로깅 시스템(Prometheus, Grafana, Loki 등) 구축 및 유지보수
   - 시스템 성능 분석 및 병목 현상 해결

인프라 엔지니어(시스템 엔지니어)의 역할은?

3. **보안 및 정책 적용**
   - 인증서 관리(cert-manager, Vault 등)
   - RBAC/네임스페이스 기반 권한 분리
   - 보안 취약점 점검 및 패치 자동화
4. **개발자 생산성 지원**
   - 개발자가 신속하게 테스트 및 배포할 수 있도록 셀프서비스 환경 제공
   - 개발자의 요구사항을 이해하고, 이를 인프라 구성으로 빠르게 반영
   - 운영 환경의 피드백을 개발 팀과 적극적으로 공유

단계 개념

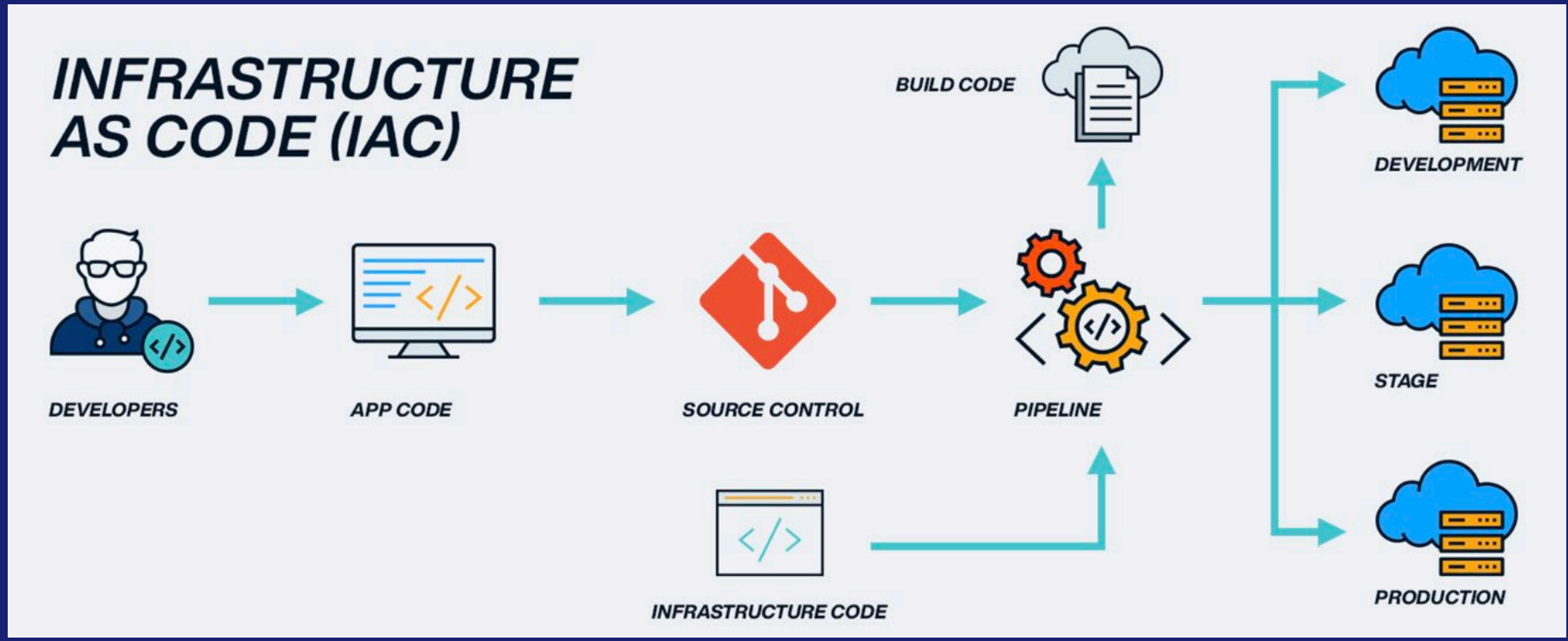
CI/CD는 다음과 같은 단계 개념을 제공한다.

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 오류**를 사전에 방지한다.



지속적 제공(CD, Continuous Delivery)

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

1. 수동 승인 또는 별도 절차를 통해 언제든 프로덕션에 릴리스 할 준비를 완료한다.
2. 지속적 제공 단계에서 추가 승인 없이 자동으로 프로덕션 환경에 배포한다.
3. 실서비스에 대한 즉각적인 피드백을 확보하여 빠른 개선 사이클을 실현한다.

CD 도구는 **회사/팀/개인** 마다 다르기 때문에, 상황에 맞는 적절한 도구를 선택한다.

대표 도구 정리

빌드 및 테스트 자동화(CI)

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

배포 파이프라인(CD)

- Argo CD
- GitLab CD
- Tekton

대표 도구 정리

인프라 자동화 및 구성 관리(쿠버네티스/오픈스택 사용가능)

- Terraform
- Ansible
- Helm (Kubernetes 패키징)

모니터링 및 피드백

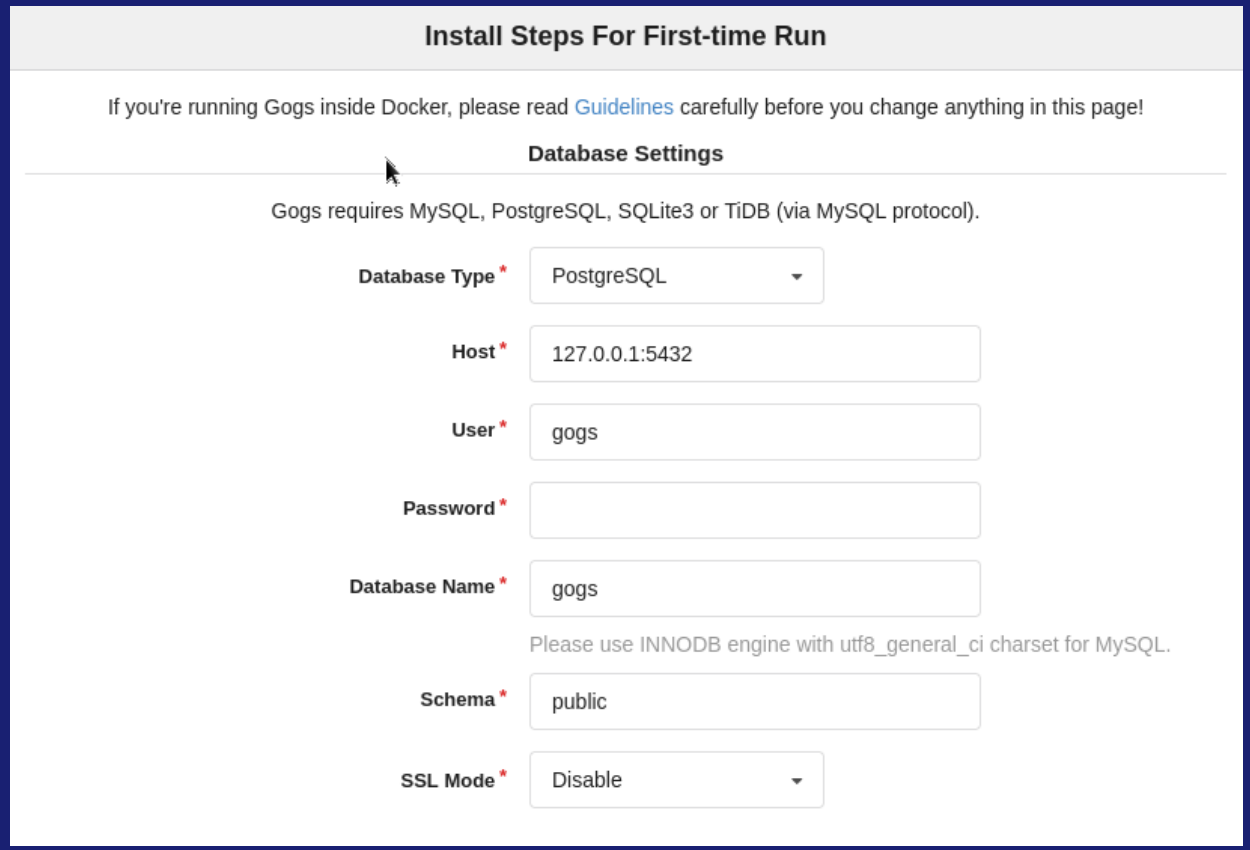
- Prometheus
- Grafana
- ELK 스택

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
```

랩


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

랩

```
[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-svsca...  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 도구**라고 보시면 된다. 테크톤은 Continuous Integration기능도 제공하나, SCM에 구성이 되어 있는 소스코드를 가져와서 이미지 빌드를 할 수 있도록 지원한다.

테크톤은 개발자에게 다음과 같은 기능 제공한다.

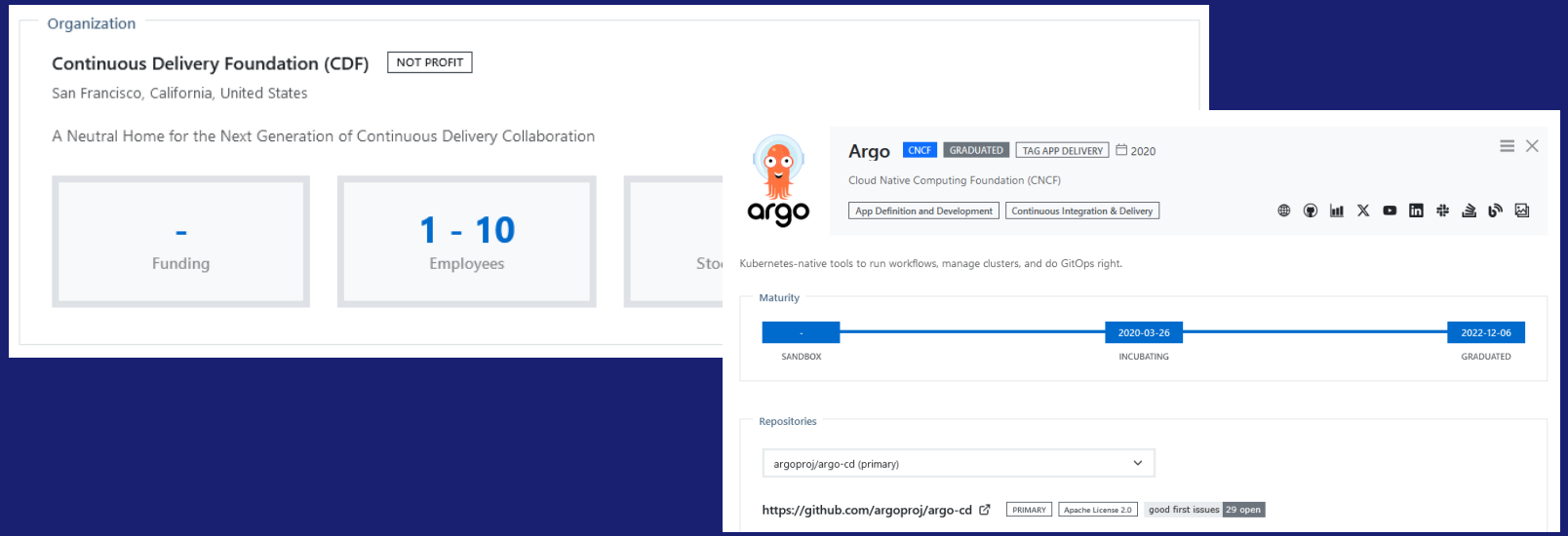
1. build
2. test
3. deploy

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

TEKTON

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

ArgoCD는 CNCF에서 GRADUATED된 프로젝트이다.



2025-07-21

TEKTON VS JENKINS

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

젠킨스 Groovy라는 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

젠킨스 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/

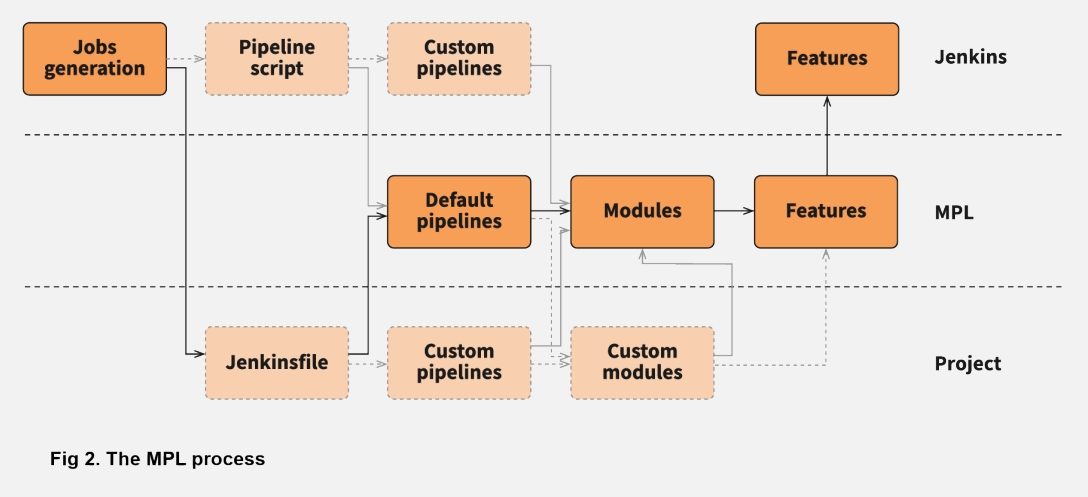
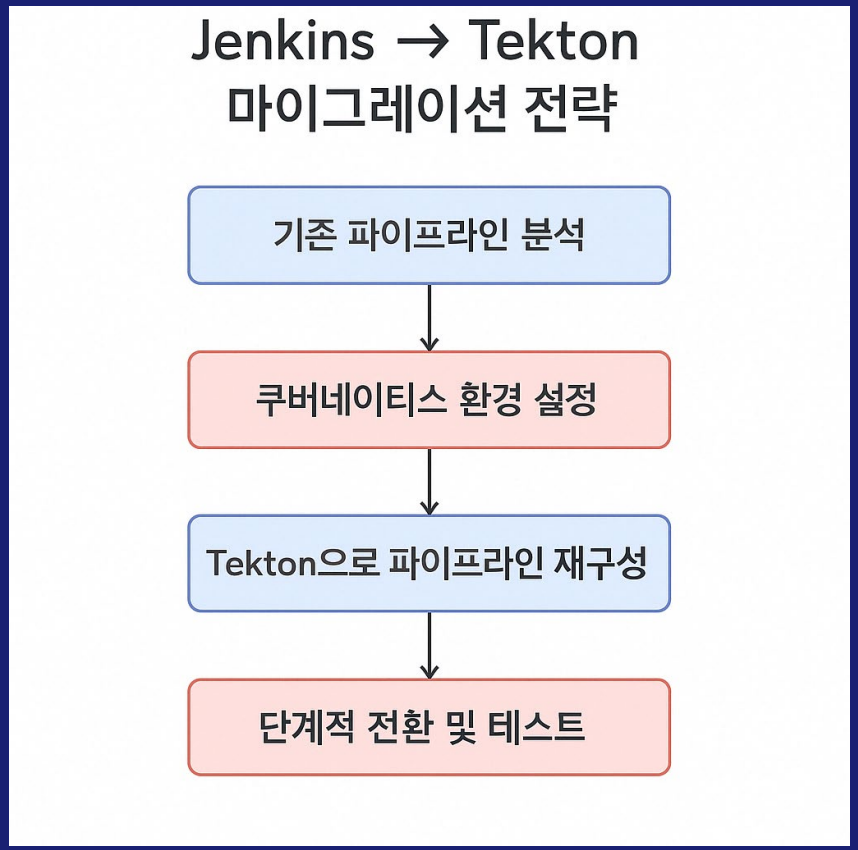
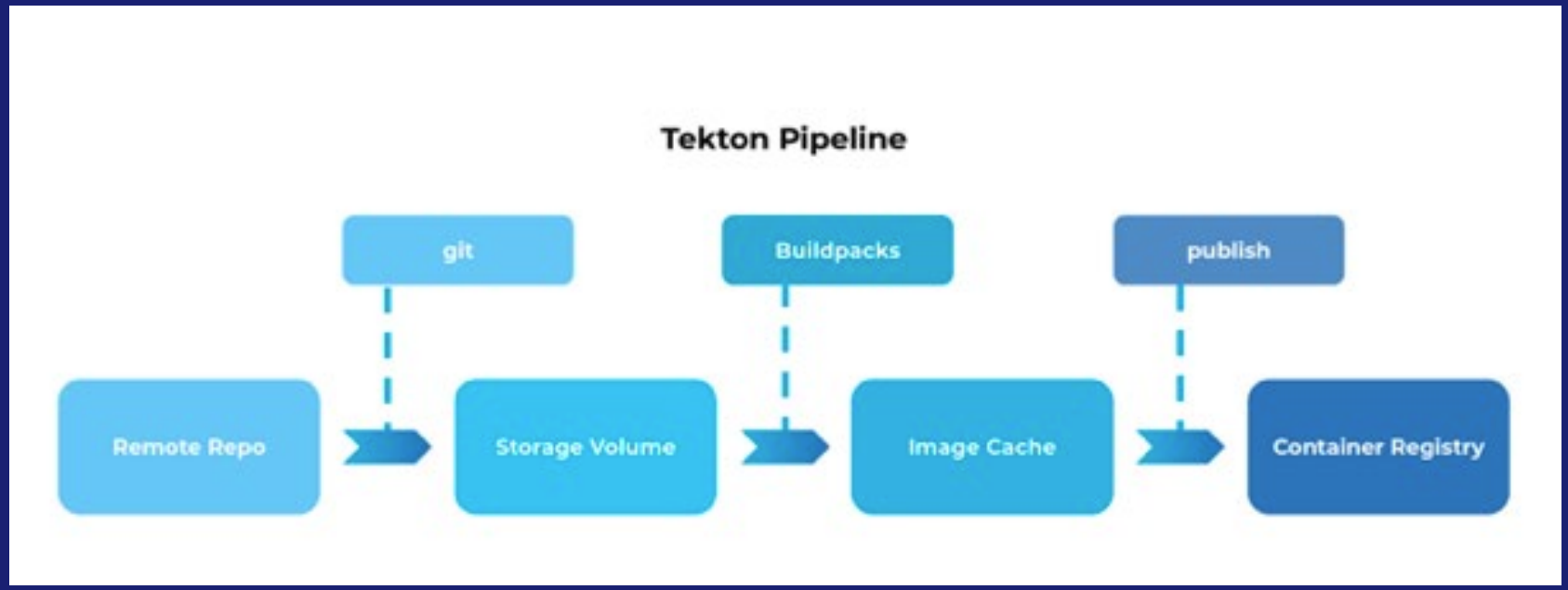


Fig 2. The MPL process

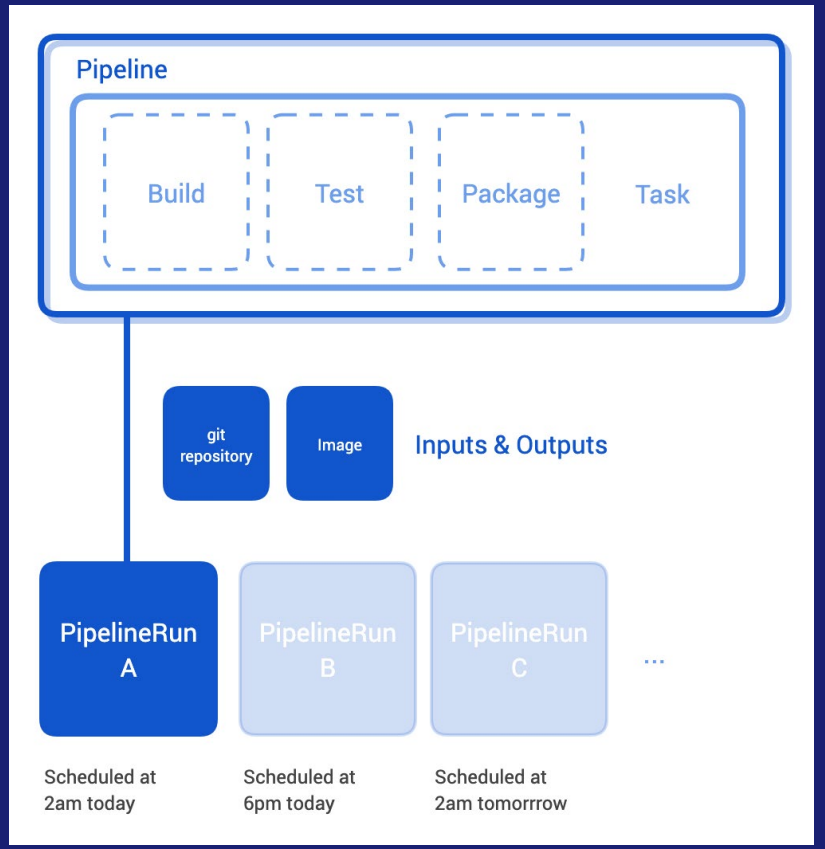
마이그레이션


Jenkins → Tekton
마이그레이션 전략
기존 파이프라인 분석
쿠버네이티스 환경 설정
Tekton으로 파이프라인 재구성
단계적 전환 및 테스트

IN KUBERNETES



TEKTON PIPELINE



클라우드 네이티브에서 CI/CD

Tekton

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.

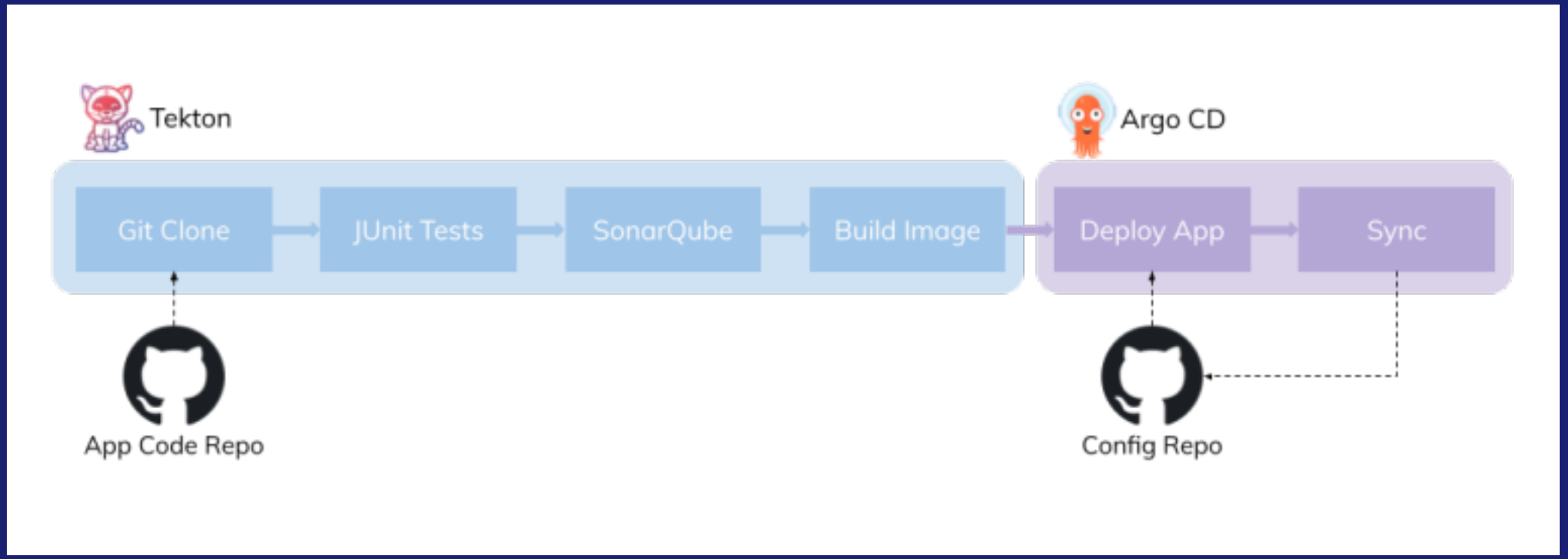
CNCF/K-Native

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

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

Tekton+ArgoCD



STEP

기본자원

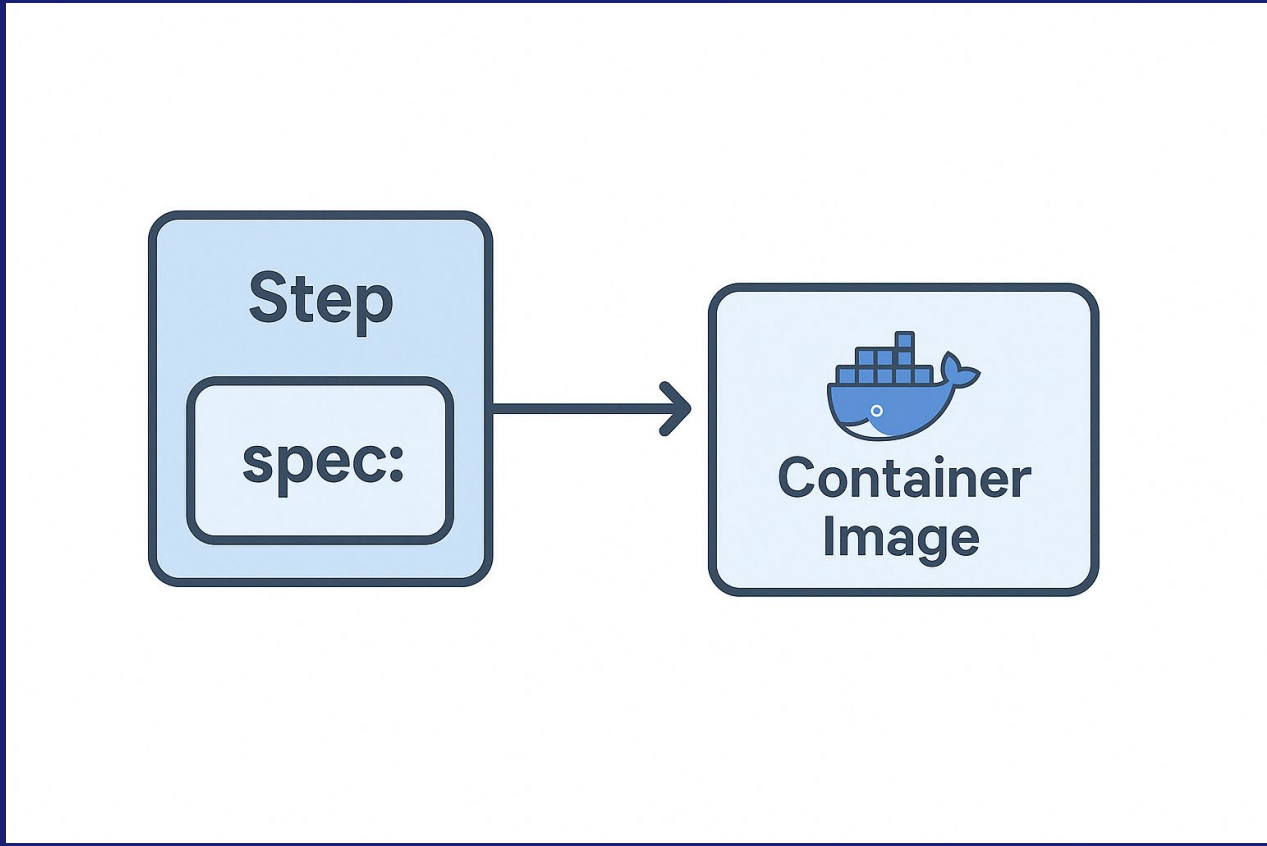
STEP

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 수행할지 명시한다. 자세한 내용은 뒤에서 더 다룬다.

```
spec:
  steps:
    - image: quay.io/centos/centos
      command:
        - /bin/bash
        - -c
        - echo "Hello World"
```

STEP



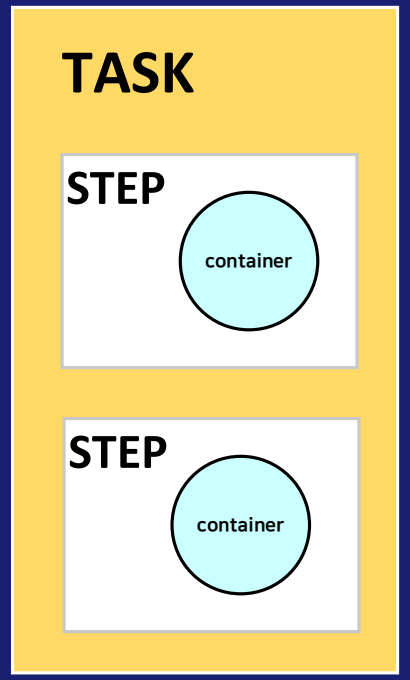
TASK, PIPELINE

기본 자원

TASK

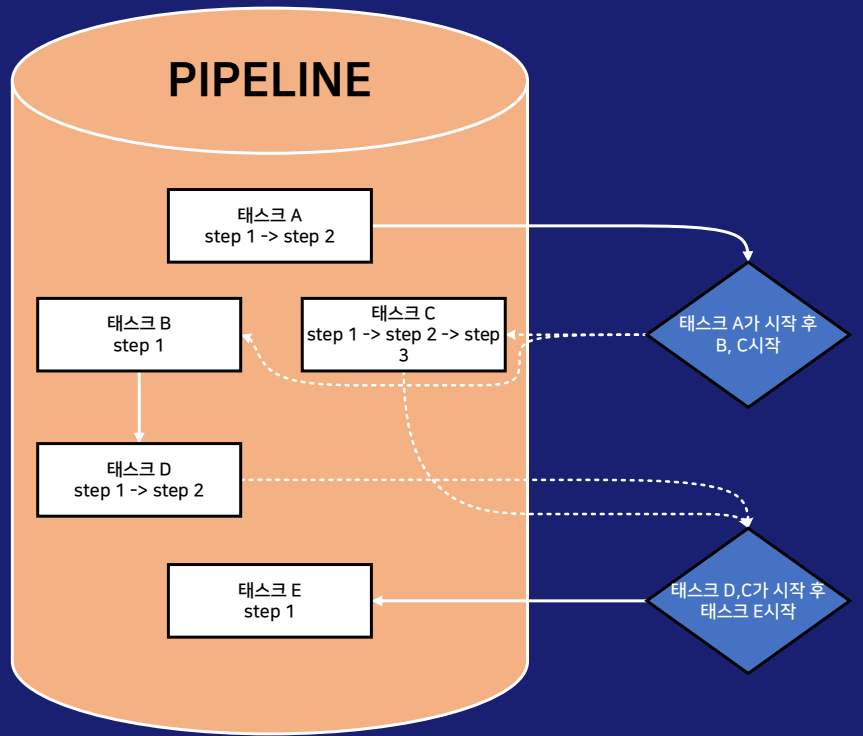
파이프 라인은 **작업의 순서를 정하는 자원**이다.

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 YAML기반으로 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 파이프라인**이며, 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
first-pipeline   14 minutes ago   first-pipeline-run-86x7v   19 seconds ago
```

기본 자원

Stepping

Task

Pipe

STEP

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

1. 소스코드 내려받기
2. 컴파일 및 명령어 수행
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                  7 seconds ago
hello                        3 minutes ago
# 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 centos이미지 기반으로 구성하였다. 이미지는 최소 한 개가 구성 및 선언이 되어야 한다.

문법은 아래와 같이 선언이 된다.

현재 연습 예제에서는 centos기반으로 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

파라메터에 기본값 설정이 필요한 경우 default라는 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
```

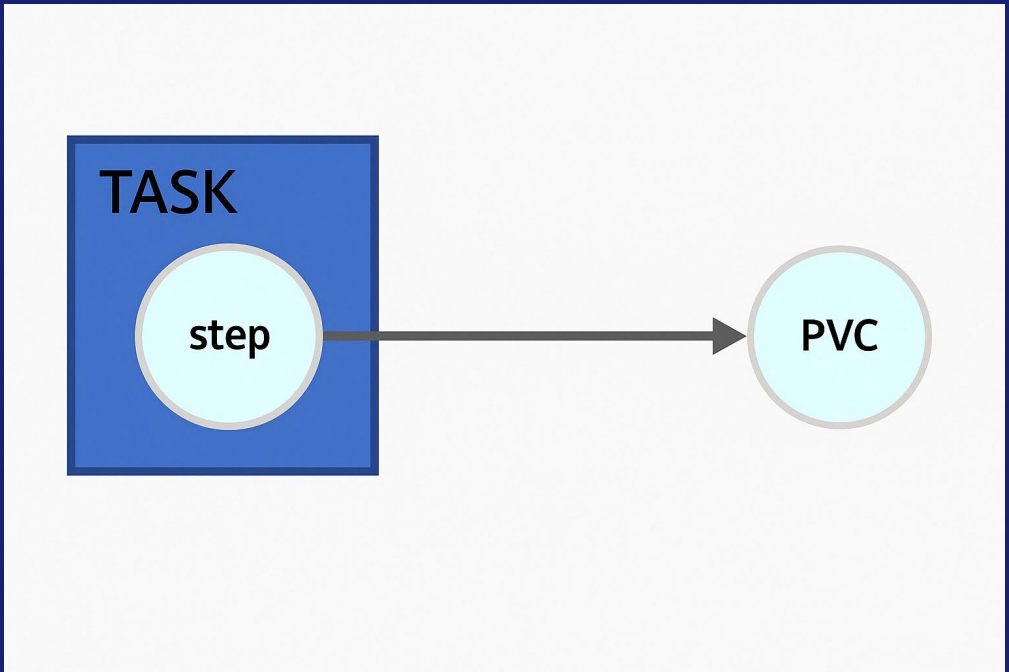
TASK DEFAULT PARAMETER

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
```

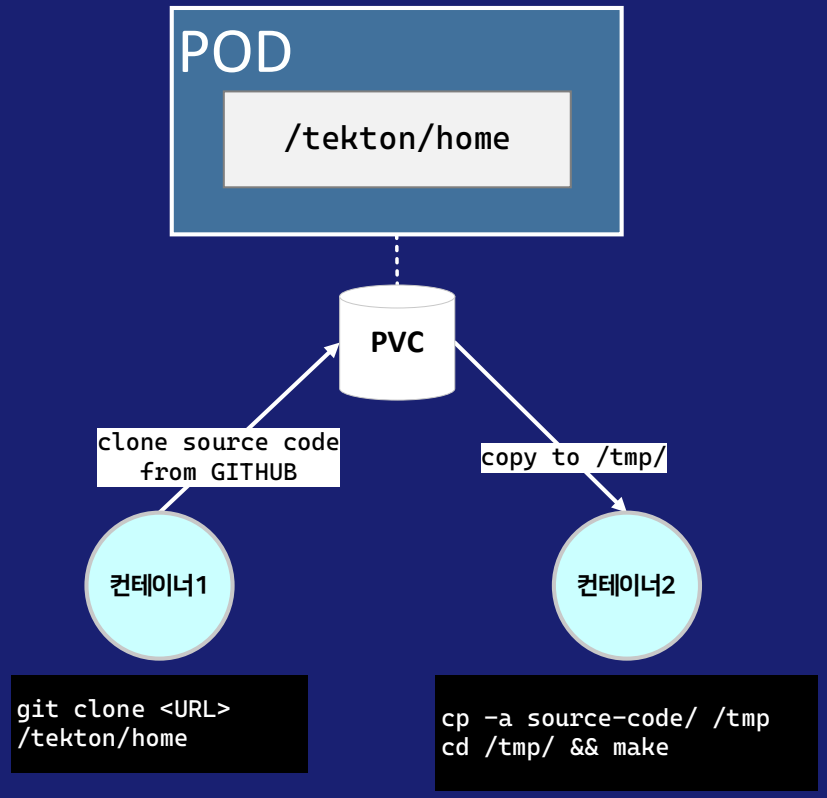
공유 데이터

테크톤이 동작하면 컨테이너가 하나의 포드(Pod)에서 동작하기 때문에, 데이터 공유가 가능하다. 데이터를 공유 하기 위해서 쿠버네티스의 PV/PVC를 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 소개
    - PPT 링크
2. 부트캠프 2차 강의(날짜 미정, 대규모 환경에서 IaC 활용)
3. 부트캠프 3차 강의(날짜 미정, IaC 및 인프라 환경에서 CPU AI 활용)