

# DevOps 이해와 적용

김 순곤

[soongon@hucloud.co.kr](mailto:soongon@hucloud.co.kr)

# 주요 내용

---

- DevOps 문화와 사례를 포함한 기본사항
  - IaC .. Infrastructure as Code
  - DevSecOps - 프로세스에 보안 추가
  - SDLC 주기 동안 몇 가지 DevOps 사례
  - 애플리케이션 컨테이너화 - Docker & Kubernetes
- 
- 다양한 도구를 통한 DevOps 사례를 단계별로 구현

# SW 개발 방법론으로의 DevOps

---

- SDLC .. Software Development Life Cycle

- 현재 문제 식별
- 계획 수립
- 요구사항 정의
- 디자인 및 프로토타이핑
- 소프트웨어 개발
- 테스트
- 배포
- 운영 및 유지관리

# SW 개발 방법론으로의 DevOps

---

- SDLC 방법론

- Waterfall
- Agile
- DevOps
- 기타
  - ❖ 빅뱅, 스파이럴, 반복, V모델

# SW 개발 방법론으로의 DevOps

---

- SDLC 방법론 베스트 프랙티스
  - 소스통제
    - ❖ Git, GitHub, GitLab ..
  - 지속적 통합
    - ❖ CI / CD
    - ❖ Jenkins, GitLab CI ..
- + 문화

# 실습 준비 사항

---

- AWS 계정
- GitHub 계정 (옵션)
- DockerHub 계정 (옵션)
- MobaXterm / Putty (for Windows)

- DevOps 시작하기
- CI/CD 및 지속적 배포 구현
- IaC 이해

# 1. DevOps 이해

# DevOps 시작하기

# DevOps 들어가기

---

- **비즈니스 목표**

- 사용자 만족도를 유지하면서 제품(서비스)을 더 빠르게 설계하고 제공

- **솔루션**

- 개발팀, 운영팀, 테스터, 보안팀 간 협업 문화 도입
  - -> DevOps 문화

# DevOps

---

- We do DevOps, We use DevOps tools
  - 개발과 운영이라는 단어의 축약형
- 기존 기업 문화와 다름
  - 사고방식, 프로세스 및 도구의 변화가 필요
- 주로
  - CD / CI
  - IaC (Infrastructure as Code)

# DevOps

---

- 2008 애자일 컨퍼런스
  - 애자일 인프라스트럭처에 대해 논의
  - Patrick Debois
- 2009 벨기에 데브옵스 데이를 통해 대중화
- 개발자와 운영 간의 장벽을 줄이는 일련의 관행
- 애자일 프로세스 (Scrum, XP) 의 확장

# DevOps 의 세가지 축

---

- 협업문화
- 프로세스
- 도구 (tools)

# DevOps 의 세가지 축

---

- 협업문화 - DevOps 핵심

- 팀이 더 이상 사일로 전문화(개발팀, 운영팀, 테스터 팀, 보안 팀..)로 더이상 구분되지 않음
- 제품에 부가가치를 부여한다는 동일한 목표로 뭉침

- 프로세스

- 애자일 방법론의 개발 프로세스를 따름
- CI (개발 워크플로)와 CD (배포 워크플로) 통합
- DevOps 프로세스는 여러 단계로 나뉨

# DevOps 의 세가지 축

---

## ● 프로세스

- 계획 및 기능 우선순위 부여
  - 개발
  - CI (Continuous integration) and CD (delivery)
  - Continuous Deployment
  - Continuous Monitoring
- 
- 위 프로세스는 프로젝트에서 반복적으로 수행됨

# DevOps 의 세가지 축

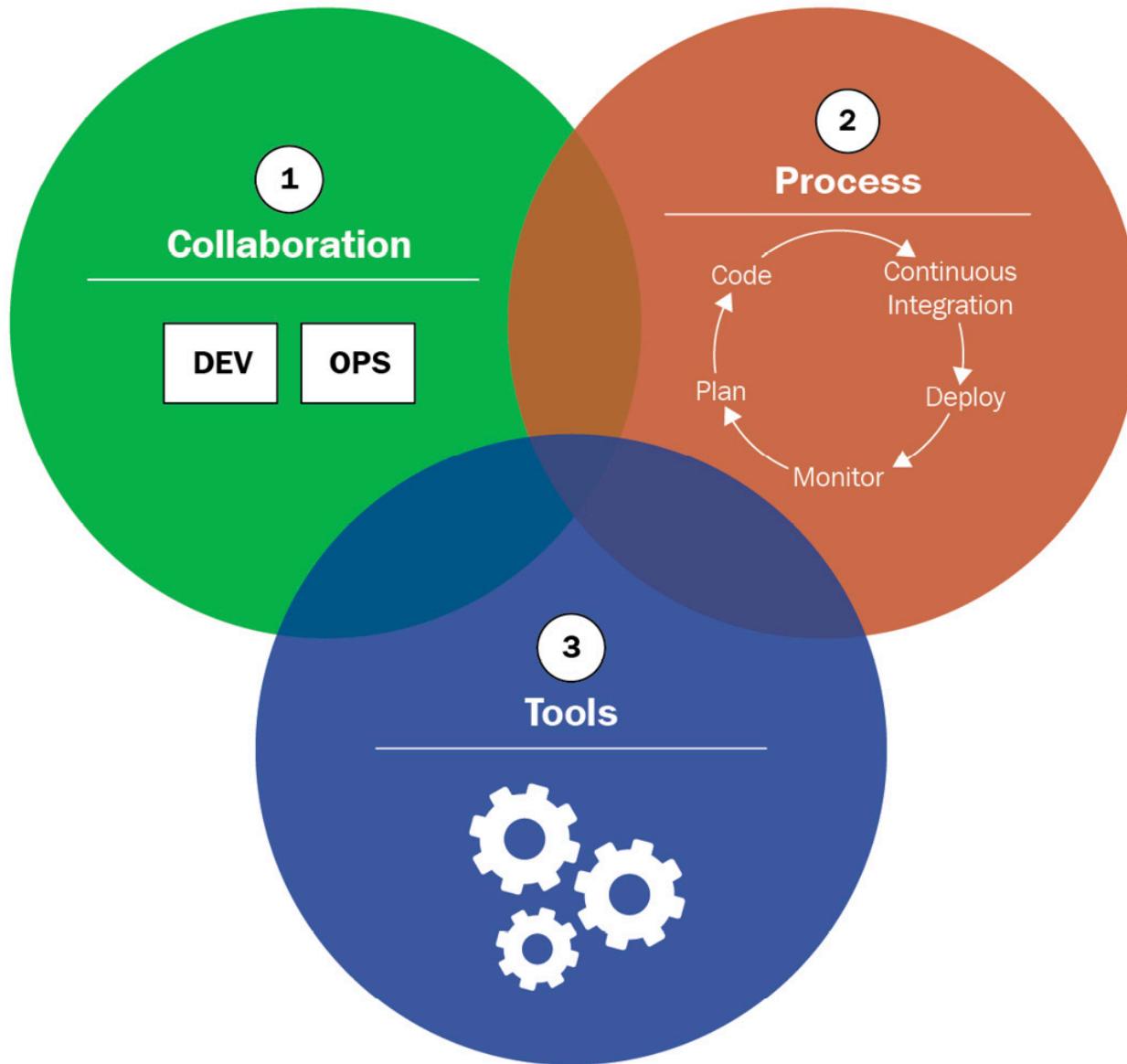
---

- 도구 (tools)

- 팀에서 사용하는 도구와 제품의 선택은 매우 중요
- Dev 와 Ops 를 모두 포함하는 도구 사용

<https://roadmap.sh/devops>

# DevOps 문화



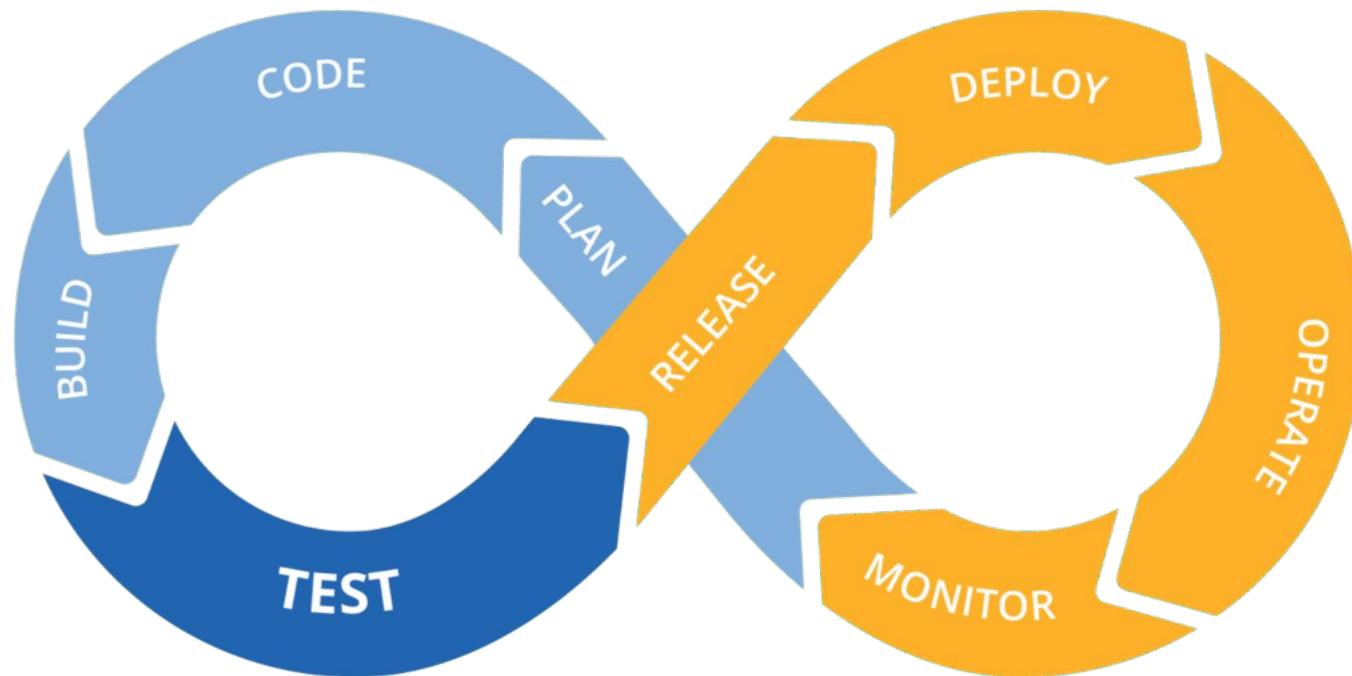
# DevOps 효과

---

- 팀의 더 나은 협업 및 의사 소통
- 생산까지의 리드 타임이 단축,
  - 성능과 사용자 만족도 향상
- IaC로 인프라 비용 절감
- 자동화로 인한 애플리케이션 오류 감소
- 새로운 기능 개발에 더 집중

# DevOps

---



# **CI/CD 및 지속적인 배포 구현**

# CI / CD

---

- 지속적인 통합, Continuous Integration (CI)
- 지속적인 전달, Continuous Delivery (CD)
- 지속적인 배포, Continuous Deployment (CD)

# 지속적 통합, CI

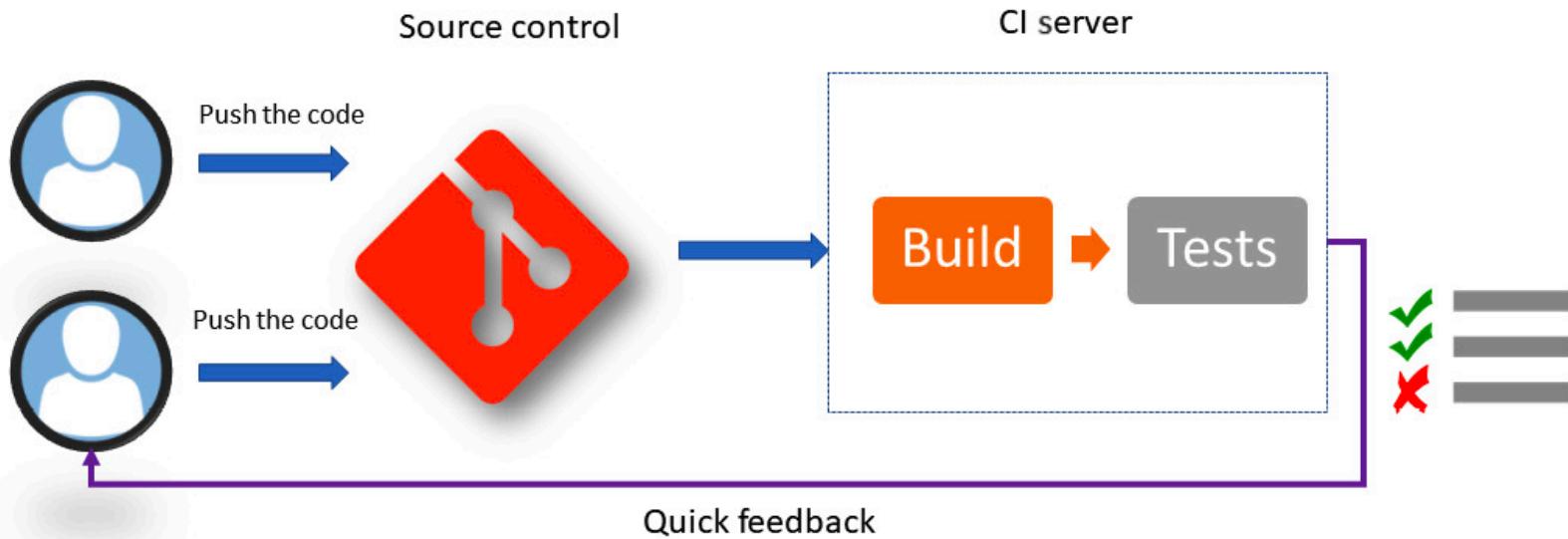
---

- 자동화된 빌드 (테스트 포함)
- CI 구현
  - SCM - Git, SVN
  - 자동 빌드툴 -
    - ❖ Jenkins, GitLab CI, TeamCity, Azure Pipeline, GitHubActions, Travis CI, AWS CodeBuild

# 지속적 통합, CI

## 커밋 통합 - CI 프로세스의 시작

- 애플리케이션 패키지 빌드 - 컴파일, 파일 변환 등
- 단위 테스트 수행
- 정적 코드와 취약점 분석 (옵션)



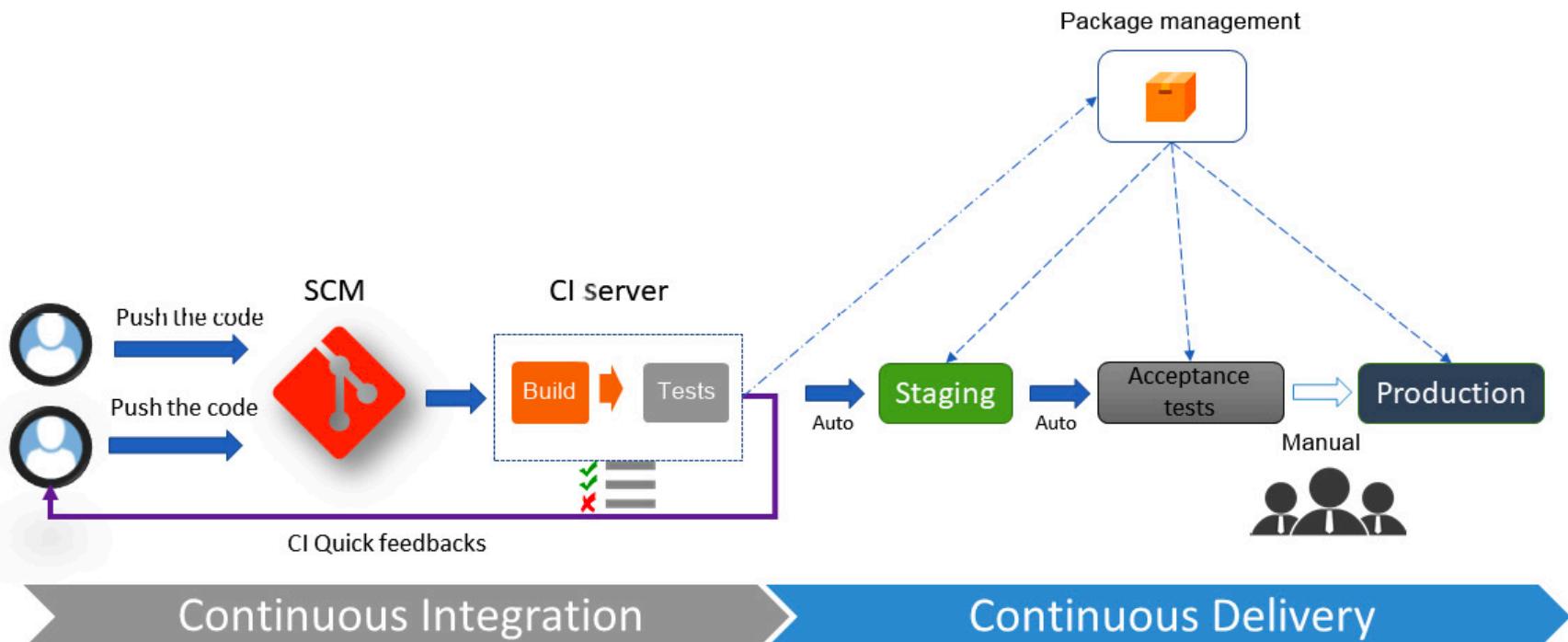
# 지속적 전달, CD

---

- 스테이징 환경으로 애플리케이션 배포
  - 지속적 통합 과정이 완료되면 수행
  - CI에 의해 패키징된 결과물로 시작
  - unzip, 서버 시작/중지, 파일복사, 설정 변경 등 ..
  - 기능 테스트와 인수 테스트 수행
- CD 구현 도구
  - 패키지 관리자 - Nexus, ProGet, Artifactory
  - 구성 관리자

# 지속적 전달, CD

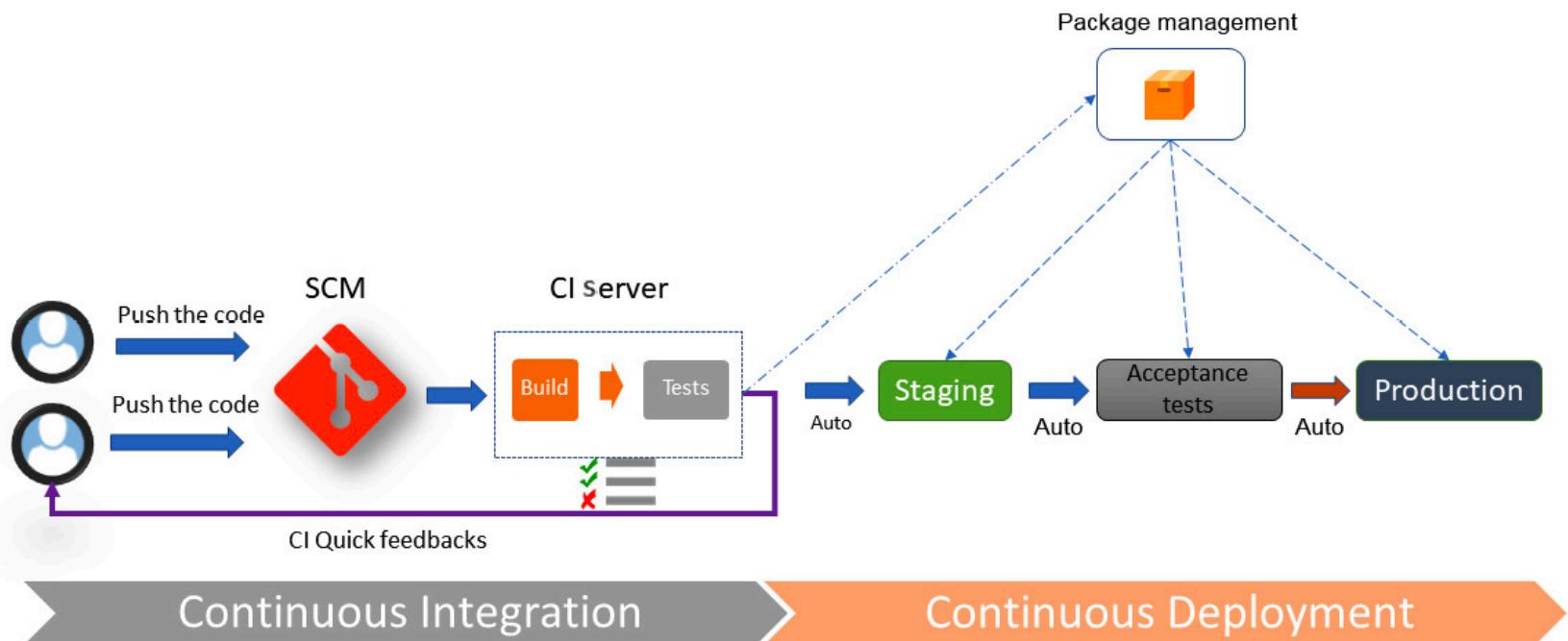
- 스테이징 환경으로 애플리케이션 배포
  - 프로덕션 환경으로의 배포 과정은 자동과 수동 선택



# 지속적 배포, CD

## ● 프로덕션 환경에 배포

- 자동으로의 배포는 거의 구현되지 않음
- 프로덕션 문제 발생 시 애플리케이션 복원 고려
- 블루-그린 프로덕션 환경 구성 가능



IaC, Infrastructure as Code 0 | 0

# IaC 란?

---

- 인프라를 구성하는 리소스를 코드로 표현
  - DevOps 문화와 클라우드 인프라와 함께 적용
  - 자동화된 인프라 프로비저닝과 변경
  - 반복가능하고 일관된 방식으로 배포를 자동화
- 인프라 구성의 표준화 -> 오류 위험을 줄여줌
- 인프라를 설명하는 코드가 버저닝되고 제어됨
- 코드가 CI / CD 파이프라인에 통합
- 인프라 변경 배포의 효율성과 비용절감

# IaC 언어와 도구

---

- 스크립팅 방식
  - CLI 툴 - bash, powershell, SDK ..
- 선언적 방식
  - Terraform, Ansible, Puppet, Chef ..
- 프로그램적 방식
  - Typescript, Java, Python 사용 가능
  - Pulumi, Terraform CDK ..

# IaC 언어와 도구

---

- 스크립팅 방식

- CLI 툴 - bash, powershell, SDK ..

- Azure CLI 와 Azure PowerShell 사용 예

```
az group create --location westeurope --resource-group MyAppResourcegroup
```

```
New-AzResourceGroup -Name MyAppResourcegroup -Location westeurope
```

# IaC 언어와 도구

- 선언적 방식

- Terraform, Ansible, Puppet, Chef ..

- Terraform 과 Ansible 의 코드 사례

```
resource "azurerm_resource_group" "myrg" {  
    name = "MyAppResourceGroup"  
    location = "West Europe"  
    tags = {  
        environment = "Bookdemo"  
    }  
}
```

```
---  
- hosts: all  
  tasks:  
    - name: install and check nginx latest version  
      apt: name=nginx state=latest  
    - name: start nginx  
      service:  
        name: nginx  
        state: started
```

# IaC 언어와 도구

## ● 프로그램적 방식

- TypeScript로 작성된 Terraform CDK 코드

```
import { Construct } from 'constructs';
import { App, TerraformStack, TerraformOutput } from 'cdktf';
import {
    ResourceGroup,
} from './gen/providers/azurerm';
class AzureRgCDK extends TerraformStack {
    constructor(scope: Construct, name: string) {
        super(scope, name);
        new AzurermProvider(this, 'azureFeature', {
            features: [{}],
        });
        const rg = new ResourceGroup(this, 'cdktf-rg', {
            name: 'MyAppResourceGroup',
            location: 'West Europe',
        });
    }
}
const app = new App();
new AzureRgCDK(app, 'azure-rg-demo');
app.synth();
```

# IaC 형태

---

- 클라우드에서의 IaC 구성 형태
  - 인프라 프로비저닝과 배포
  - 서버 구성과 템플리팅
  - 컨테이너화
  - 쿠버네티스에서의 구성 및 배포

# IaC 형태

---

- 인프라 프로비저닝과 배포

- 인프라를 구성하는 리소스를 인스턴스화 하는 작업
- PaaS, 웹앱, 서비스 리소스, VNet, 서브넷, 라우팅 테이블, 방화벽, 네트워크 ..
- 프로비저닝 도구
  - ❖ Terraform, ARM 템플릿, AWS CloudFormation

# IaC 형태

---

- 서버 구성과 템플리팅

- Configuration Management
  - 가상머신 구성에 관련된 것
    - ❖ 디렉토리, 디스크 마운팅
    - ❖ 네트워크 구성(방화벽, 프록시 ..)
    - ❖ 미들웨어 설치
  - Configuration manager 툴
    - ❖ Ansible, Chef, Puppet, SaltStack ..

# IaC 형태

## ● 서버 구성과 템플리팅

- Templating - 서버 템플리트 작성
- 주요 툴 - Aminator(Netflix), HashiCorp Packer

```
{  
  "builders": [{  
      "type": "azure-arm",  
      "os_type": "Linux",  
      "image_publisher": "Canonical",  
      "image_offer": "UbuntuServer",  
      "image_sku": "16.04-LTS",  
      "managed_image_resource_group_name": "demoBook",  
      "managed_image_name": "SampleUbuntuImage",  
      "location": "West Europe",  
      "vm_size": "Standard_DS2_v2"  
    }],  
  "provisioners": [  
    {  
      "execute_command": "chmod +x {{ .Path }}; {{ .Vars }} sudo -E sh '{{ .Path }}'",  
      "inline": [  

```

Packer 파일 코드 일부

# IaC 형태

---

## ● 컨테이너화

- 애플리케이션을 VM 대신 컨테이너에 배포
- Docker
- Docker 이미지는 Dockerfile에 코드로 구성
- 컨테이너는 VM과 달리 변경이 불가 - 실행 중 수정 불가

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y nginx
ENTRYPOINT ["/usr/sbin/nginx","-g","daemon off;"]
EXPOSE 80
```

Dockerfile 예

# IaC 형태

## ● 쿠버네티스에서의 구성 및 배포

- 컨테이너 오케스트레이션 기술
- IaC 로 구현
  - ❖ 컨테이너 배포 방식,
  - ❖ 네트워크 아키텍처 (로드밸런서, 포트)
  - ❖ 볼륨관리, 민감한 정보 보호
  - ❖ YAML 파일에 설명

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-demo
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

yaml 파일 예

# IaC 관행

---

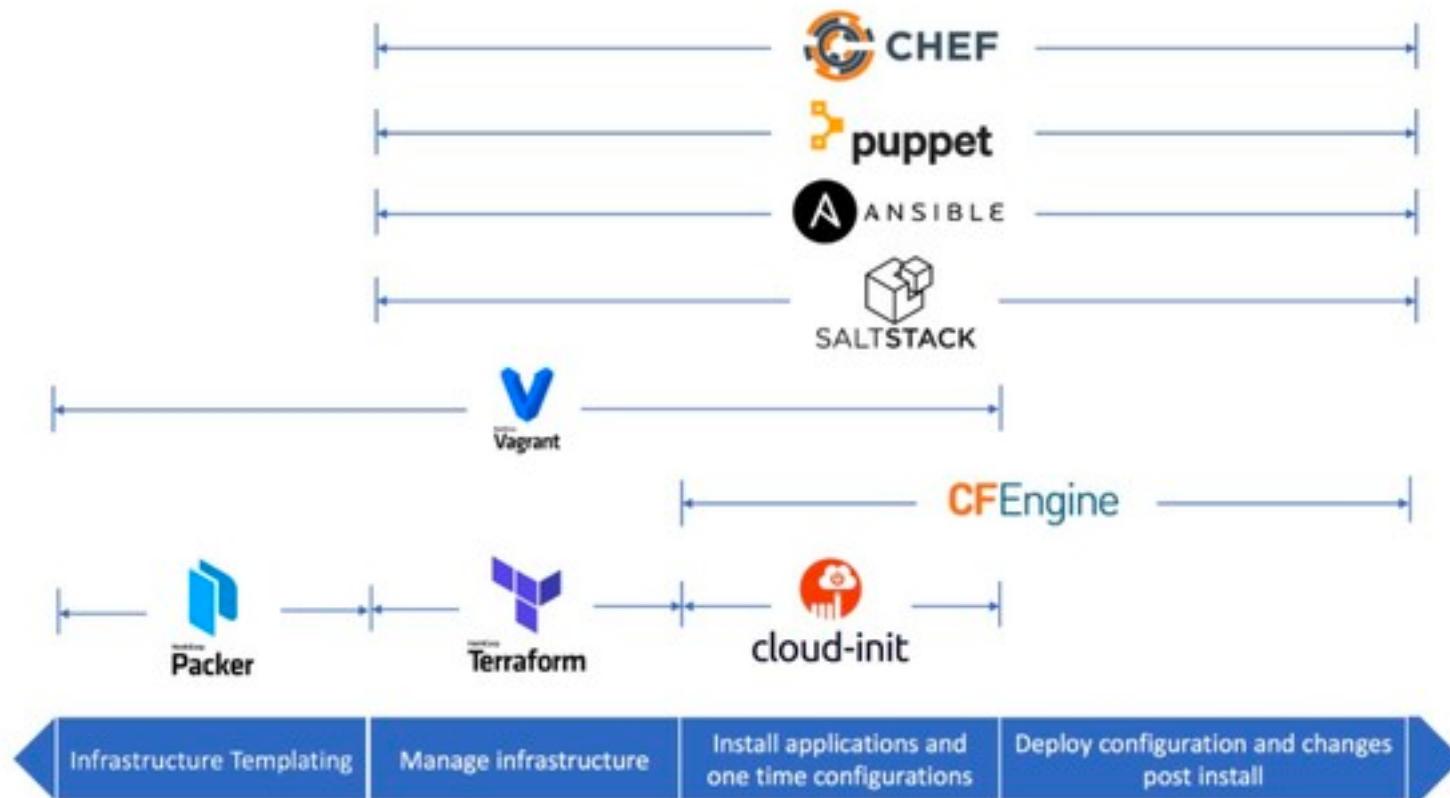
- 좋은 명명법 사용
- 불필요한 주석으로 코드에 과부하 주지 않기
- 작은 기능 사용
- 에러 처리 구현

# IaC Practices

---

- 코드에서 모든것이 자동화
- 코드는 소스 관리자에 있어야 함
- 인프라 코드는 애플리케이션 코드와 같이 있어야 함
- 역할에 따라 디렉토리 분리
- CI / CD 프로세스에 통합
- 코드는 매번 실행 시 결과가 동일해야 함
- 인프라 코드는 문서역할을 할수 있어야 함
- 코드는 모듈식으로 작성

# 각 단계별 적용되는 IaC 툴



- Terraform 설치
- Terraform 구성
- Terraform 스크립트 작성
- 배포용 Terraform 실행
- 명령줄 옵션 사용

## 2. Terraform - 클라우드 인프라 프로비저닝

# Terraform

---

- by HashiCorp
  - 멀티 플랫폼 (Windows, Mac, Linux 에 설치 가능)
  - 다양한 클라우드 프로바이더 지원 (AWS, GCP, Azure ..)
  - 인프라 변경 전 미리보기 허용
  - 리소스 종속성 고려하여 작업을 병렬화 가능
- HCL, HashiCorp Configuration Language
  - 오픈소스, 커맨드라인 툴
  - 선언적이며 사용하기 쉬움

# 테라폼 역사

---

- 2014년 7월 by HashiCorp
  - HashiCorp - Vagrant, Packer, Vault ..
  - Atlas 제품의 일부로 Terraform Enterprise 호스팅 서비스
- 주요 지원 클라우드
  - AWS, GCP, MS Azure
  - 다른 프로바이더는 커뮤니티에서 개발 및 지원
- 테라폼 코드는 Go 언어로 작성됨

# 테라폼 설치

---

- 매뉴얼 설치

- <https://www.terraform.io/downloads>

## Download Terraform

macOS

Windows

Linux

FreeBSD

OpenBSD

Solaris

---

# 테라폼 기본 개념

---

- **프로비저닝**
  - 프로세스나 서비스를 실행하기 위한 준비단계
- **프로바이더**
  - 테라폼과 외부 서비스를 연결해 주는 모듈
  - 예) AWS 프로바이더
- **리소스**
  - 특정 프로바이더가 제공해주는 조작 가능한 대상의 최소 단위
- **Plan**
  - .tf 파일의 내용을 실제로 적용 가능한지 확인하는 작업
- **Apply**
  - 테라폼 프로젝트 디렉토리 아래의 모든 .tf 파일의 내용대로 리소스를 생성, 수정, 삭제하는 일

# Terraform 마무리하며

---

- 테라폼 대체 도구
  - AWS CloudFormation
  - OpenStack 용 Heat
  - Chef
  - Puppet
- Immutable Infra 원칙에 적합
  - 컨테이너 환경에 적합
- Vault 내장

Git 소개 및 주요 CLI 명령어

Git 프로세스

### 3. Git - 소스코드 관리

# 소스코드 관리

---

## ● 문제점

- 내 코드를 팀원과 공유하는 방법
- 내 코드 업데이트 버전 관리 방법
- 내 코드의 변경사항을 추적하는 방법
- 내 코드 또는 그 일부의 이전 상태를 검색하는 방법

# VCS, Version Control System

---

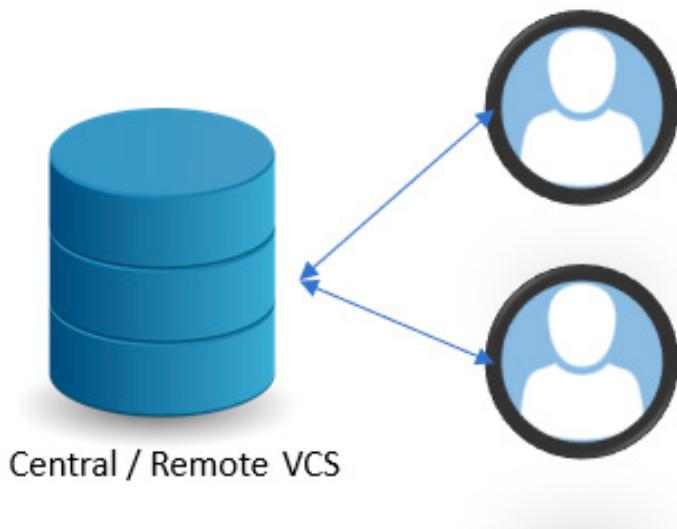
- VCS의 주요 목표

- 개발자 코드의 공동 작업 허용
- 코드 검색
- 코드 버전 관리
- 코드 변경 추적

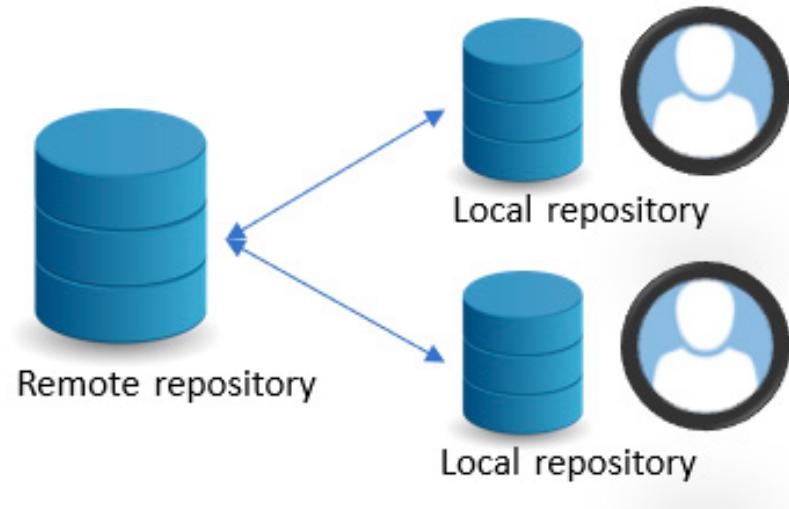
Agile 과 DevOps 문화로 VCS 사용이 필수가 됨

# VCS 사실상 표준 - Git

- 분산 소스 제어 방식 - Git



중앙 집중식

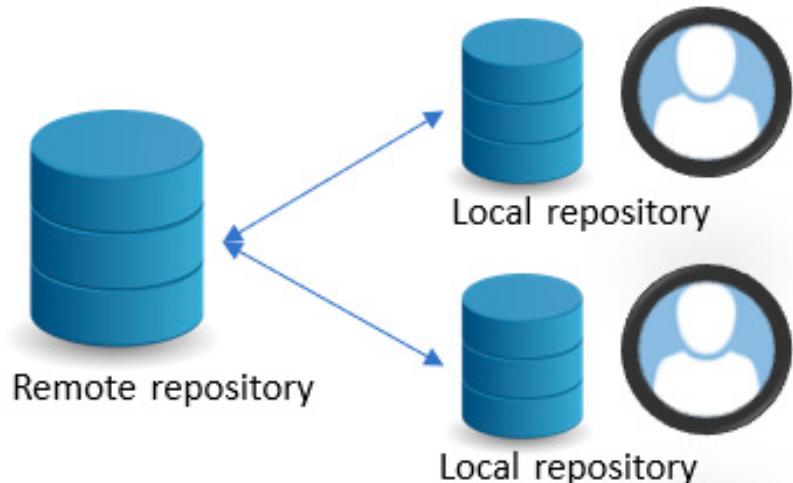


분산 소스 제어

# VCS 사실상 표준 - Git

- 2005년 by Linus Tovalds

- 오픈소스 크로스 플랫폼
- 로컬 시스템에 설치
- 원격 서버에 설치



- 원격 클라우드 리포지토리

- GitHub, GitLab, AWS CodeDeploy

**분산 소스 제어**

- GUI 도구

- GitHub Desktop, Sourcetree, GitKraken ..

# Git 설치

---

- Git은 로컬에 반드시 설치
  - Windows
  - Mac
  - Linux

# Git 주요 용어

---

- Repository:
- Clone:
- Commit:
- Branch:
- Merge:

# Git 주요 용어 (계속)

---

- Checkout:
- Fetch:
- Pull:
- Push:
- Pull request (PR):

# Git 프로세스

---

- 첫번째 개발자는 로컬 저장소에 코드를 커밋하고 원격 저장소로 푸쉬
- 두 번째 개발자는 원격 저장소에 푸쉬된 코드를 가져옴
- 두 번째 개발자는 이 코드를 업데이트하고 커밋한 다음 새 버전의 코드를 원격 저장소에 푸쉬
- 마지막으로 첫번째 개발자는 로컬 저장소에서 코드의 마지막 버전을 검색

CI / CD 원칙

Jenkins 사용

## 4. CI / CD

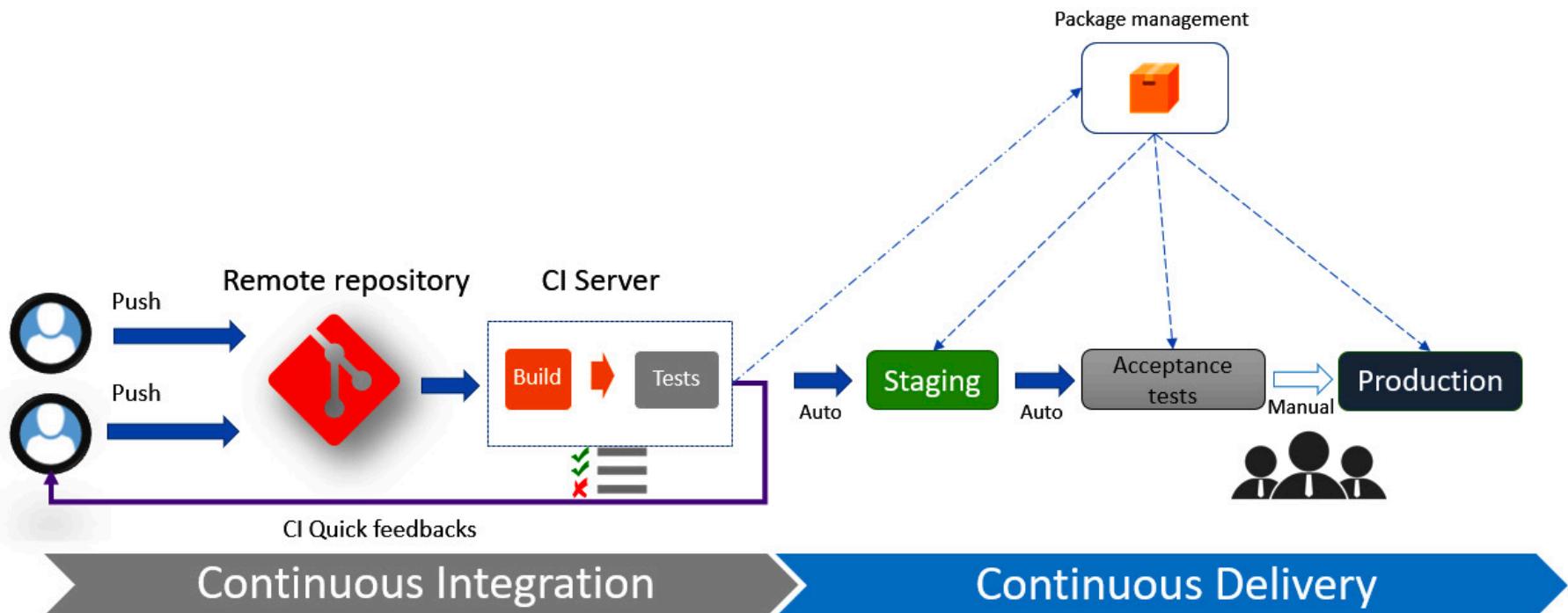
# CI / CD

---

- 지속적 통합과 지속적 전달
  - DevOps 문화의 가장 중요한 부분 중 하나
- CI
  - 팀의 모든 구성원에게 코드의 일관성과 품질에 대한 신속한 피드백을 제공하는 프로세스
- CD
  - 여러 단계(또는 환경)에서 애플리케이션을 배포하는 프로세스 자동화

# CI / CD 원칙

- CI / CD 워크플로



# CI / CD 파이프라인 설정 도구

---

- SCV
- 패키지 매니저
- CI 서버
- Configuration 매니저

Dev 와 Ops 팀이 함께 작업해야 효과적

# CI / CD 파이프라인 설정 도구

---

## ● 패키지 매니저

- 개발 라이브러리, 도구 및 소프트웨어를 중앙 집중화, 공유
  - ❖ Central Repository
- 패키지를 추적, 업데이트, 설치 및 제거 가능
- Java - Maven, Gradle
- Node(JavaScript) - npm
- Python - pypi
- .NET - NuGet

# CI / CD 파이프라인 설정 도구

---

## ● Nexus

- <https://www.sonatype.com/>
- 사내에서 운영하는 패키지 매니저
- 설치형이며, 퍼블릭 패키지 매니저 미러링
- 내부망이나 방화벽으로 외부 패키지 매니저 사용 불가시



# CI / CD 파이프라인 설정 도구

## ● Nexus 설치 및 사용

- Docker로 설치 - <https://hub.docker.com/r/sonatype/nexus3>

**Repositories** Manage repositories

	Name ↑	Type
	maven-central	proxy
	maven-public	group
	maven-releases	hosted
	maven-snapshots	hosted
	nuget-group	group
	nuget-hosted	hosted
	nuget.org-proxy	proxy

**Create repository** 1

**Repositories** / Select Recipe

Recipe ↑
apt (hosted)
apt (proxy)
bower (group)
bower (hosted)
docker (group)
docker (hosted)
docker (proxy) 2
gitlfs (hosted)
go (group)
go (proxy)
maven2 (group)
maven2 (hosted)
maven2 (proxy)
npm (group)
npm (hosted)
npm (proxy)

# CI / CD 파이프라인 설정 도구

---

- CI 서버 - Jenkins

- 빌드 서버라고 함
- Java로 개발된 오픈소스
- Jobs, Plugins (1500개 이상의 플러그인)
- 설치형이며, 주요 클라우드 서비스에서 지원



Jenkins

# CI / CD 파이프라인 설정 도구

---

- CI 서버 - GitLab CI

- Jenkins 에 대응하는 또 다른 도구 - GitLab CI
  - ❖ 소스코드 관리자 (like GitHub)
  - ❖ CI/CD 파이프라인 관리자 (like Jenkins)
  - ❖ 프로젝트 관리용 보드

<https://about.gitlab.com/features/>



- Git, GitHub
  - Jenkins
  - Maven
  - Tomcat
- 
- AWS EC2



## 실습 1. CI / CD 파이프라인

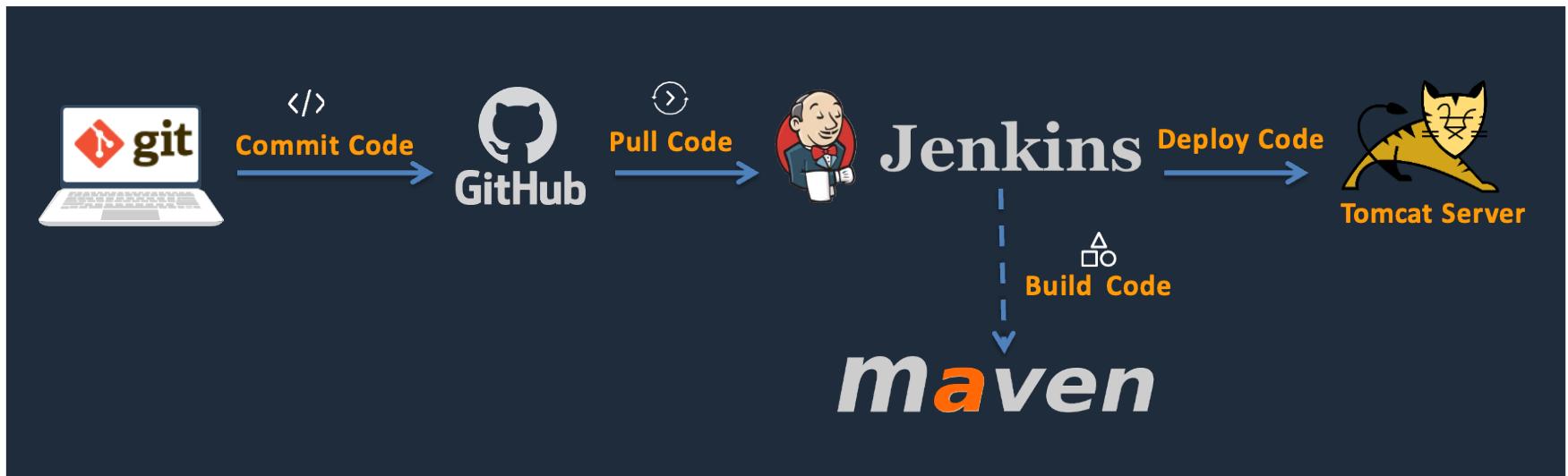
# Tomcat 서버에 빌드 후 디플로이

---

- CI / CD 셋업 - GitHub, Jenkins, Maven 그리고 Tomcat
  - 젠킨스 셋업
  - 메이븐과 깃 셋업
  - 톰캣 서버 셋업
  - 젠킨스와 GitHub, Maven, Tomcat Server 연동
  - 젠킨스 Job 생성
  - 디플로이 테스트

# 톰캣 서버에 배포

## 최종 목표



# Jenkins 서버 설치 및 셋업

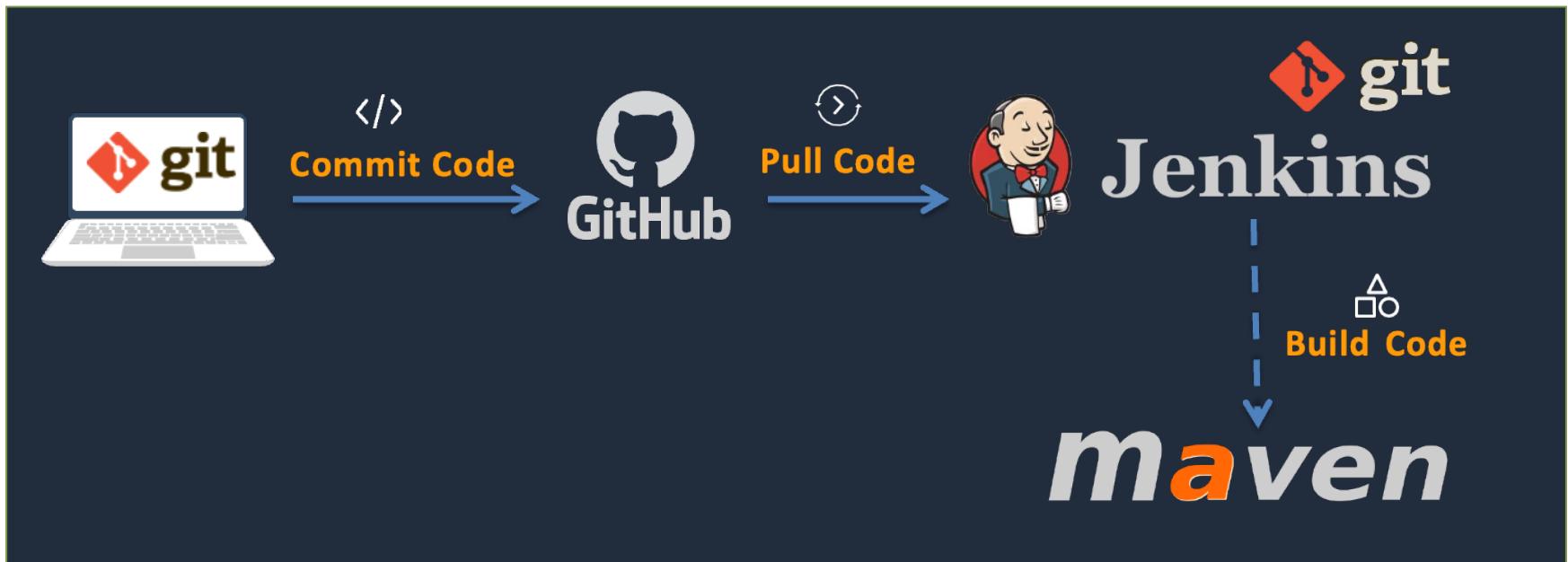
---

- AWS EC2 리눅스 서버 셋업
- 자바 설치 (JDK 11)
- 젠킨스 설치
- 젠킨스 서버 스타트
- 젠킨스 웹 UI - 8080 포트로 접속



Jenkins

# 코드 빌드



# jenkins와 깃(Git) 연동

---

- 젠킨스에 Git 설치
- 젠킨스 GUI에 GitHub 플러그인 설치
- 젠킨스 GUI에 Git 설정



# jenkins와 maven 연동

- 젠킨스 서버에 메이븐 셋업
- 환경변수 설정
  - JAVA\_HOME, M2, M2\_HOME
- 메이븐 플러그인 설정
- 메이븐과 자바 설정

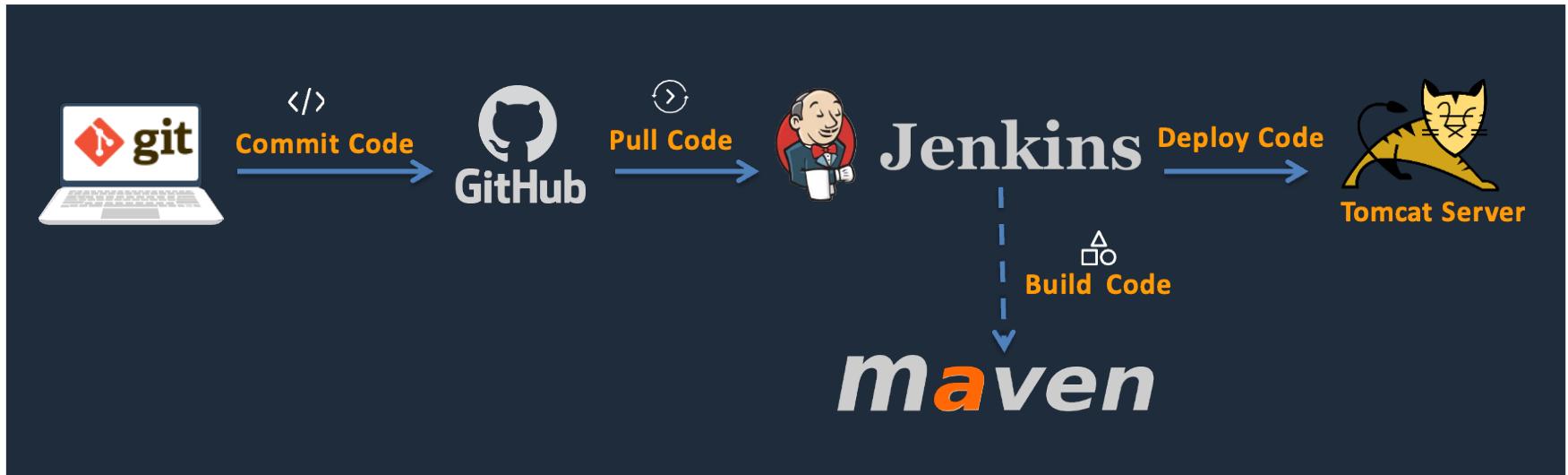


Jenkins

Build Code  


maven

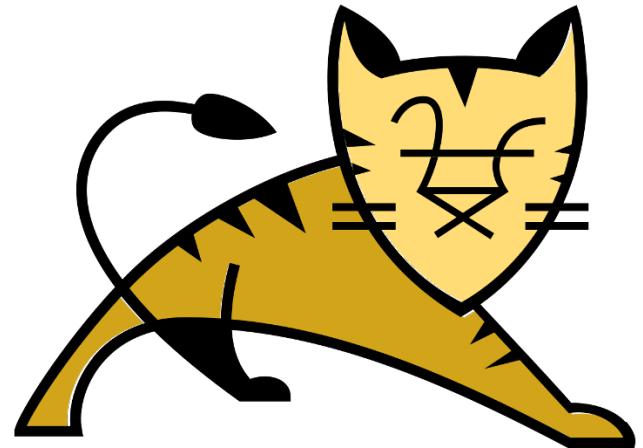
# 톰캣 서버에 배포



# 톰캣 서버 설정

---

- AWS EC2 리눅스 서버 설정
- 자바 설치
- 톰캣 다운로드 및 설치
- 톰캣 서버 스타트
- 8080 포트로 톰캣 관리자 화면 접속



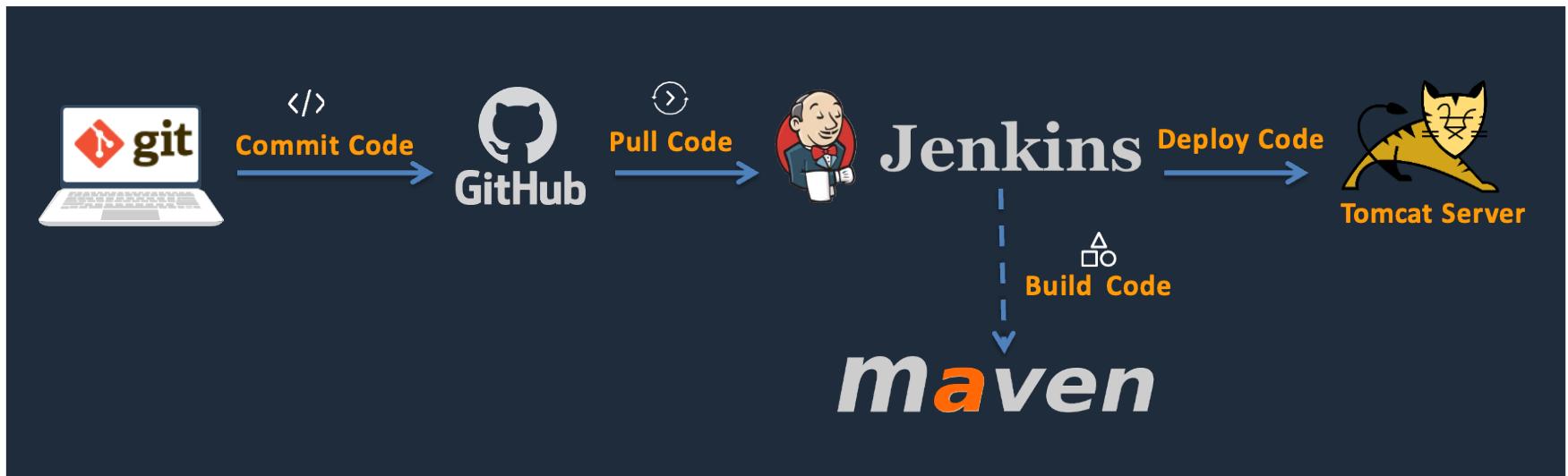
# jenkins와 톰캣 연동

- “Deploy to container” 플러그인 설치
- 톰캣 서버 Credentials 설정



# 톰캣 서버에 배포

## 최종 목표



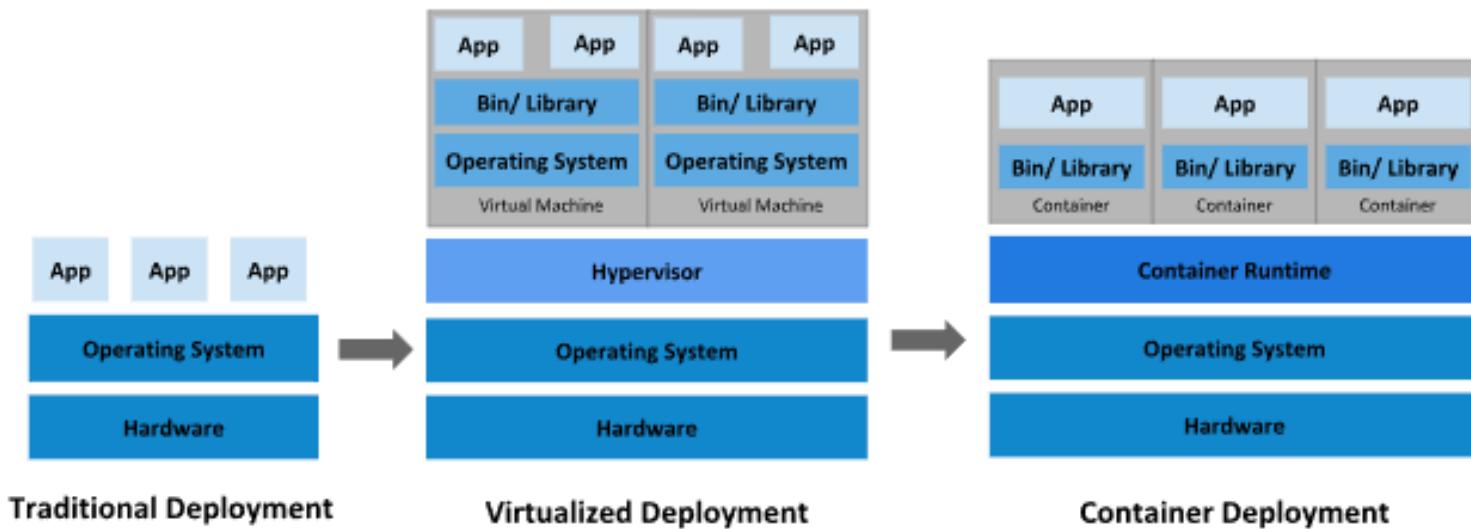
- 도커 설치
- 도커파일 작성
- 로컬머신에서 컨테이너 빌드 및 실행
- Docker Hub 에 이미지 푸시
- 명령줄 도구로 도커 사용
- 참고 : Podman

## 5. Docker – 애플리케이션 컨테이너화

# 컨테이너 란?

## 컨테이너 - 가상화

- 마이크로 서비스에서 서버 애플리케이션까지
- 코드, 바이너리, 라이브러리와 설정 까지 패키징
  - 컨테이너 이미지
- 컨테이너 이미지는 Docker 나 Podman 에 의해 실행
  - 컨테이너 가 된다.



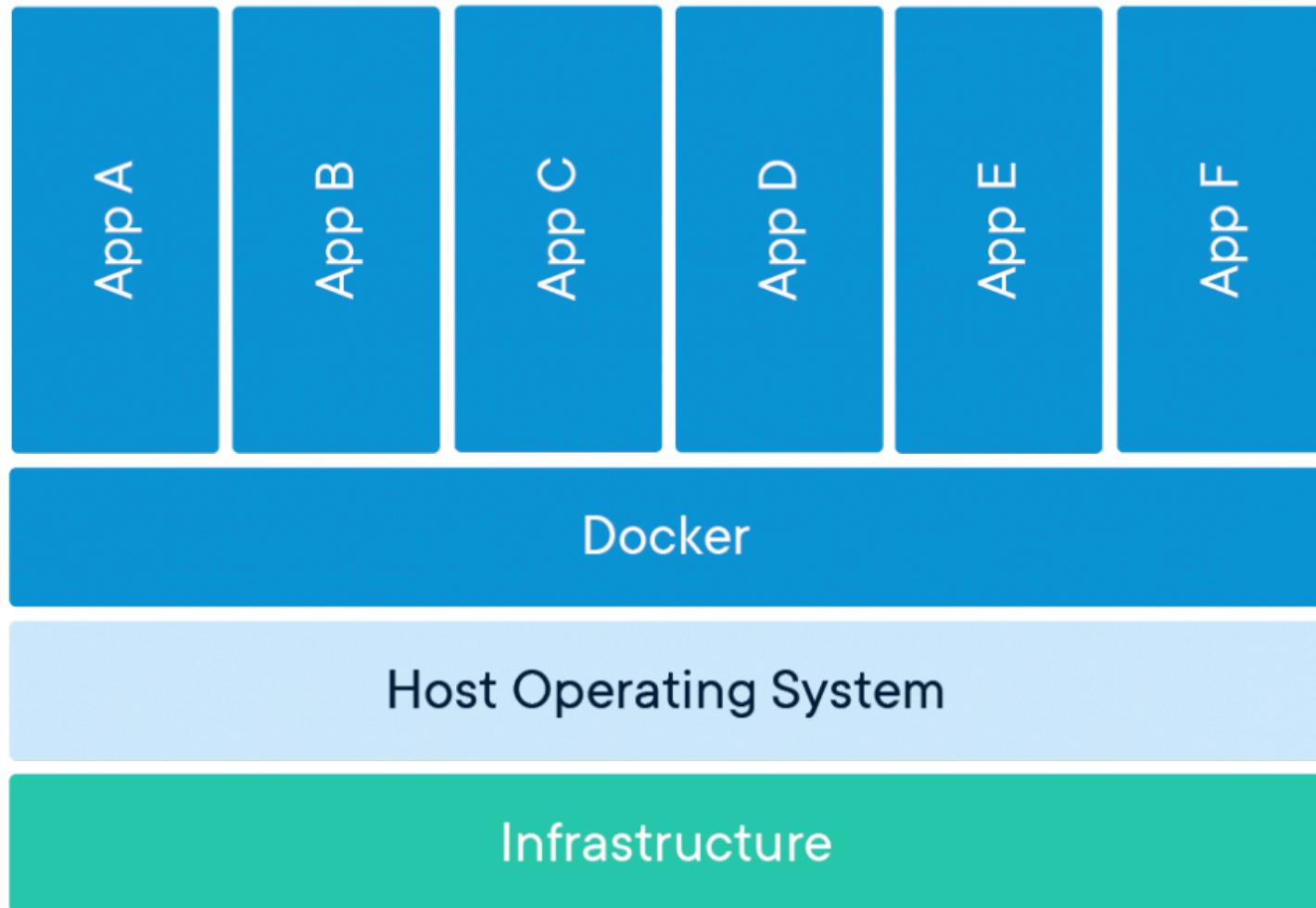
# 컨테이너 란?

---

- 컨테이너화, Containerization
  - 동일한 소프트웨어 실행(환경 및 설정)을 보장
    - ❖ 개발팀 - 애플리케이션 로직에만 집중
    - ❖ 운영팀 - 애플리케이션 버전과 설정에서 자유로움
  - 시스템 오버헤드가 적다. lightweight
    - ❖ 하드웨어 레벨 가상화(VM)가 아닌 OS-레벨 가상화 사용

# Containerized Applications

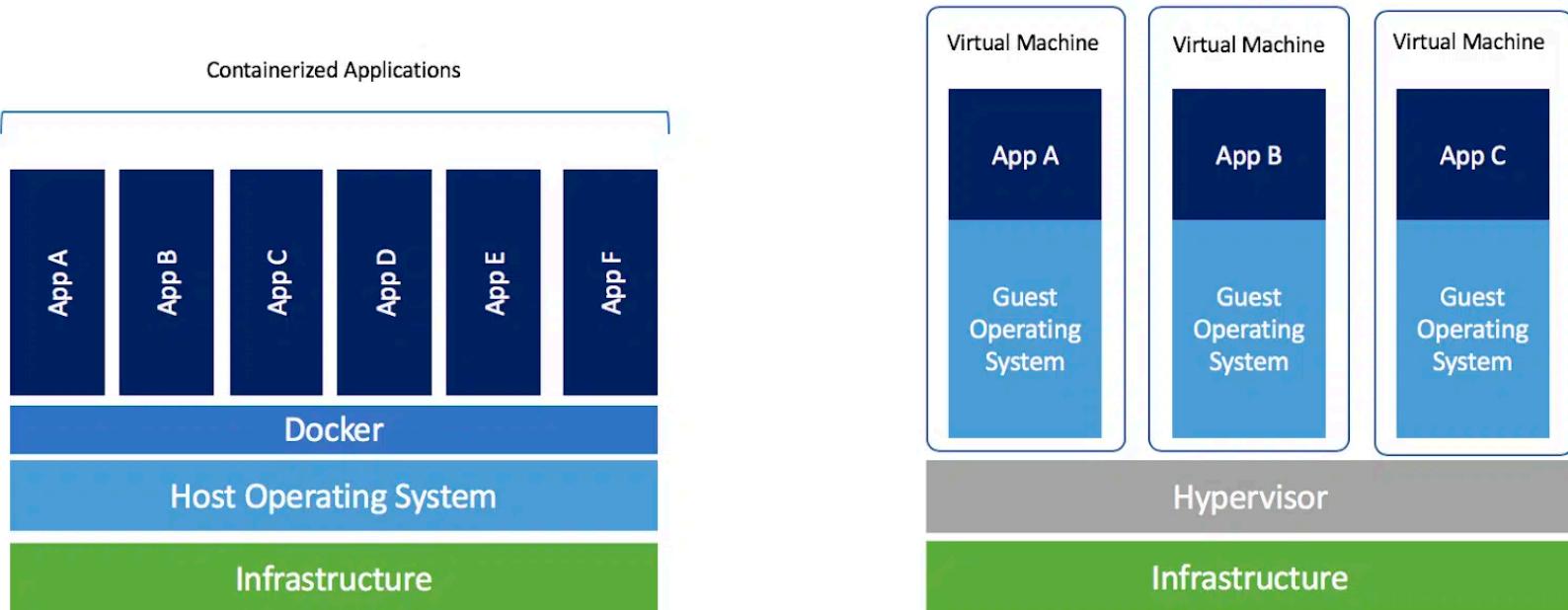
---



# 애플리케이션 컨테이너화

## 도커, Docker

- 2013년 오픈소스가 된 컨테이너화 도구
- 애플리케이션을 호스트 시스템과 격리
- 이식 가능



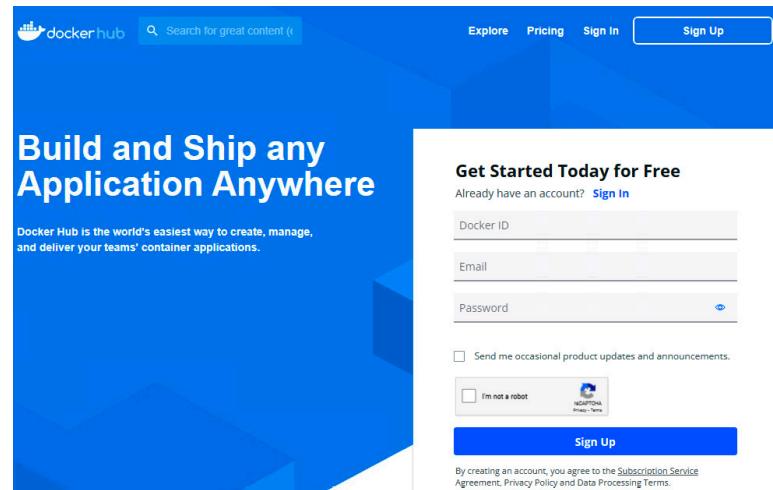
# 도커 설치

---

- 플랫폼 독립적
  - Windows, Linux, MacOS
  - AWS, Azure ..
- 도커 필수 요소
  - Docker Client - 명령줄 작업 지원
  - Docker daemon - 도커 엔진
  - Docker Registry - public (Docker Hub), private

# 도커 설치

## ● 도커 허브에 계정 만들기



## ● 도커 설치

- 로컬에 설치
- EC2에 설치

A screenshot of the Docker Docs website, specifically the "Install Docker Desktop for Windows" page. The left sidebar shows navigation links for Docker Desktop, Overview, Mac, Windows, and other sections like User manual, Networking, and Logs and troubleshooting. The main content area has a title "Install Docker Desktop on Windows" with a note about estimated reading time (9 minutes). It includes a yellow box with a warning about the Docker Desktop terms and a red box highlighting the "Download Docker Desktop for Windows" button, which is also outlined in red in the screenshot.



### Docker Desktop for Mac

A native application using the macOS sandbox security model which delivers all Docker tools to your Mac.



### Docker Desktop for Windows

A native Windows application which delivers all Docker tools to your Windows computer.

# 도커 주요 구성

---

- **도커 이미지**
  - Dockerfile 텍스트 파일에 바이너리와 애플리케이션 저장
- **도커 레지스트리**
  - 도커 이미지를 공유하기 위한 중앙 레포지토리
  - public / private
- **컨테이너**
  - 도커 이미지에서 실행되는 인스턴스
- **볼륨**
  - 호스트OS(컨테이너 외부)에 물리적으로 위치한 공간
  - 영구저장 허용

# Dockerfile

- Docker 이미지를 구축하기 위한 단계별 지침 포함
  - Apache 웹서버와 웹애플리케이션 포함된 예

```
<html>
  <body>
    <h1>Welcome to my new app</h1>
    This page is test for my demo Dockerfile.<br />
    Enjoy ...
  </body>
</html>
```

```
FROM httpd:latest
COPY index.html /usr/local/apache2/htdocs/
```

# 로컬 머신에서 컨테이너 빌드 및 실행

---

- 도커 실행 절차

- Dockerfile 에서 Docker 이미지 빌드

```
FROM httpd:latest  
COPY index.html /usr/local/apache2/htdocs/
```

- 빌드된 이미지에서 로컬로 새 컨테이너 인스턴스화

```
docker build -t demowas:v1 .
```

- 컨테이너화 된 애플리케이션 테스트

```
docker images
```

```
docker run -d --name demoapp -p 8080:80 demowas:v1
```

```
docker ps
```

# 참고 : Podman 이란?

---

- 리눅스 시스템에서 컨테이너 엔진

- OCI, Open Container Initiative 준수
- 컨테이너, 이미지 실행, 관리, 배포
- daemon-less
- docker 와 CLI command 가 동일(거의 100%)
  - ❖ docker 명령어를 podman 으로 별칭지정 사용
  - ❖ Docker Hub와 [quay.io](https://quay.io) 로 부터 컨테이너 이미지 pull/push 가능

```
$ alias docker=podman
```

- ❖ Docker Hub와 [quay.io](https://quay.io) 로 부터 컨테이너 이미지 pull/push 가능

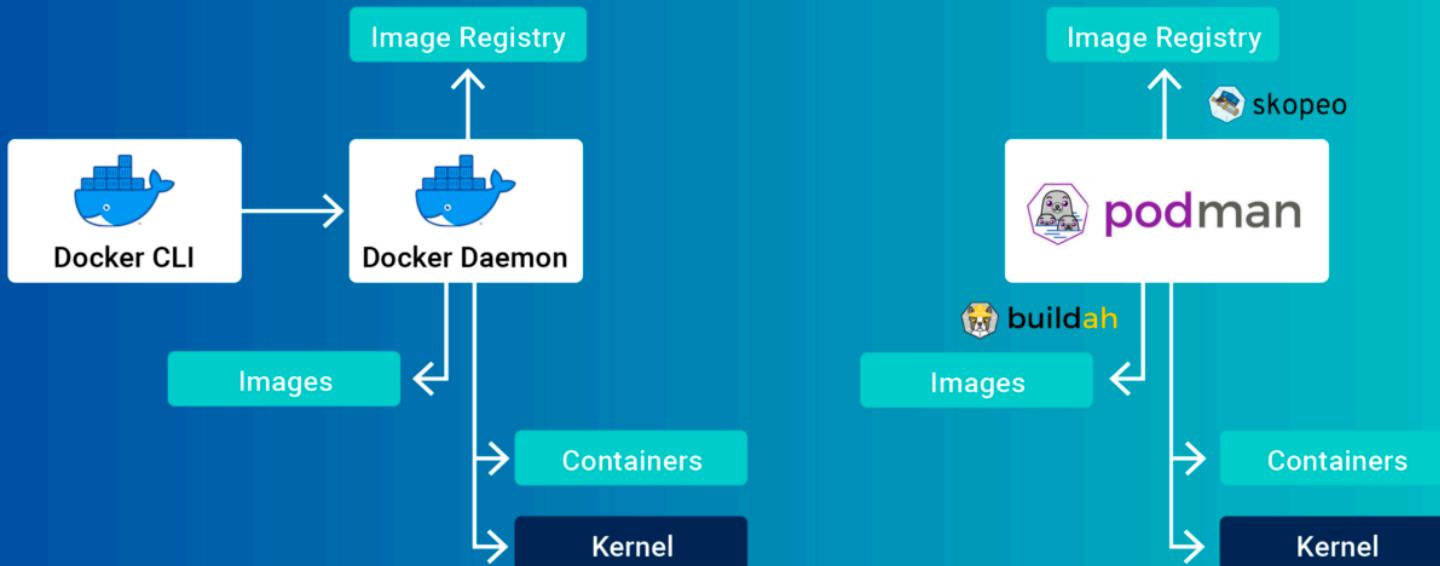
# Why Podman?

---

- Daemonless Architecture
  - Docker는 docker daemon 이 서버 역할을 수행
    - ❖ 도커 데몬이 죽으면 자식 프로세스인 컨테이너에 영향
  - Podman 은 각각의 컨테이너가 독립된 프로세스로 실행
- Rootless Container
  - 현재 접속중인 user/group 권한으로 생성/실행
  - 도커도 최근 Rootless container 지원

# Why Podman?

## Docker vs Podman



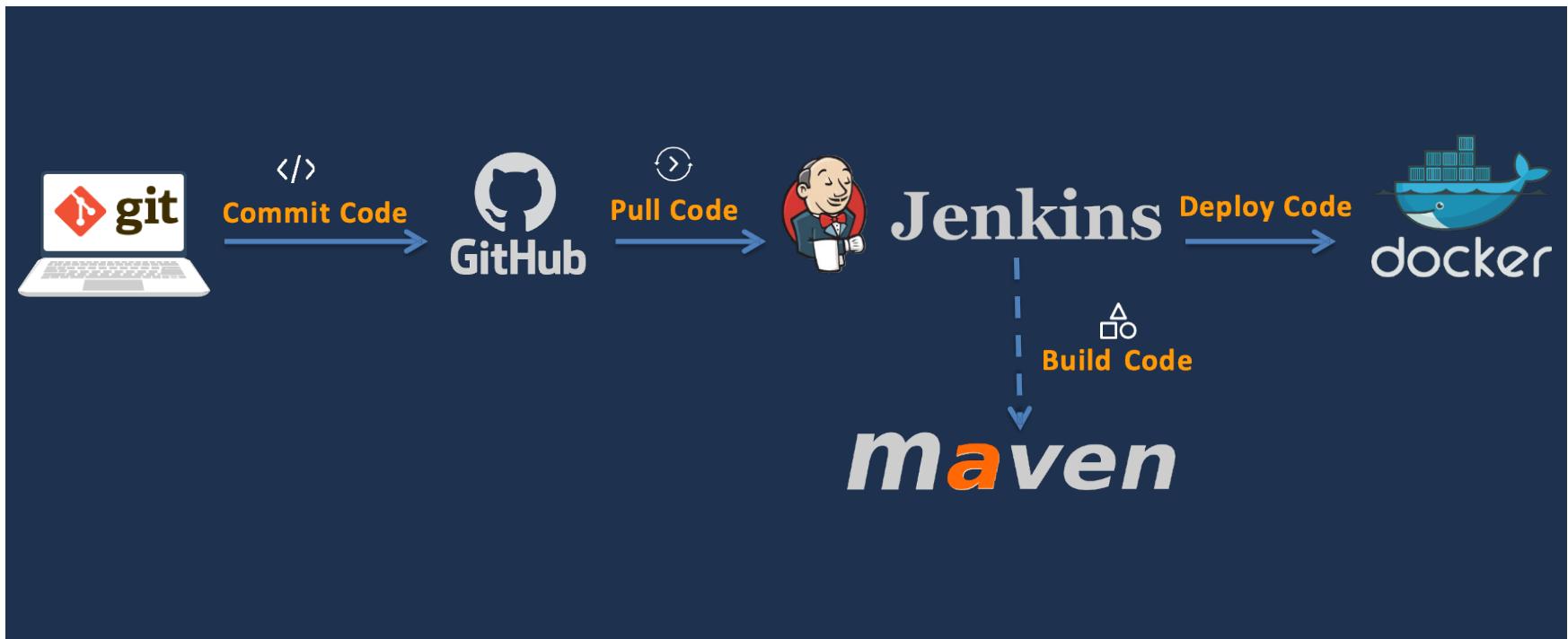
ti&m

Docker 서버 설정

Dockerfile

## 실습 2. CI / CD 파이프라인 도커와 통합

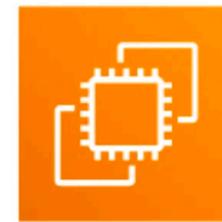
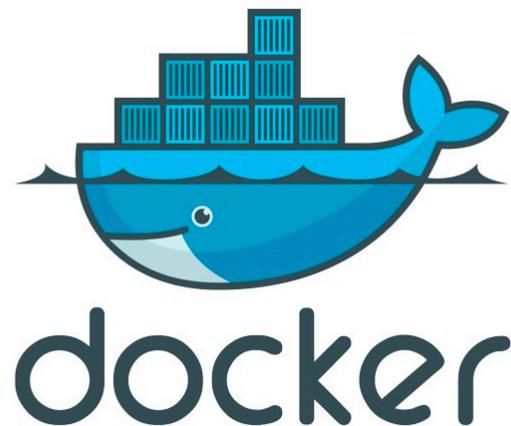
# 컨테이너에 배포



# 도커 호스트 설정

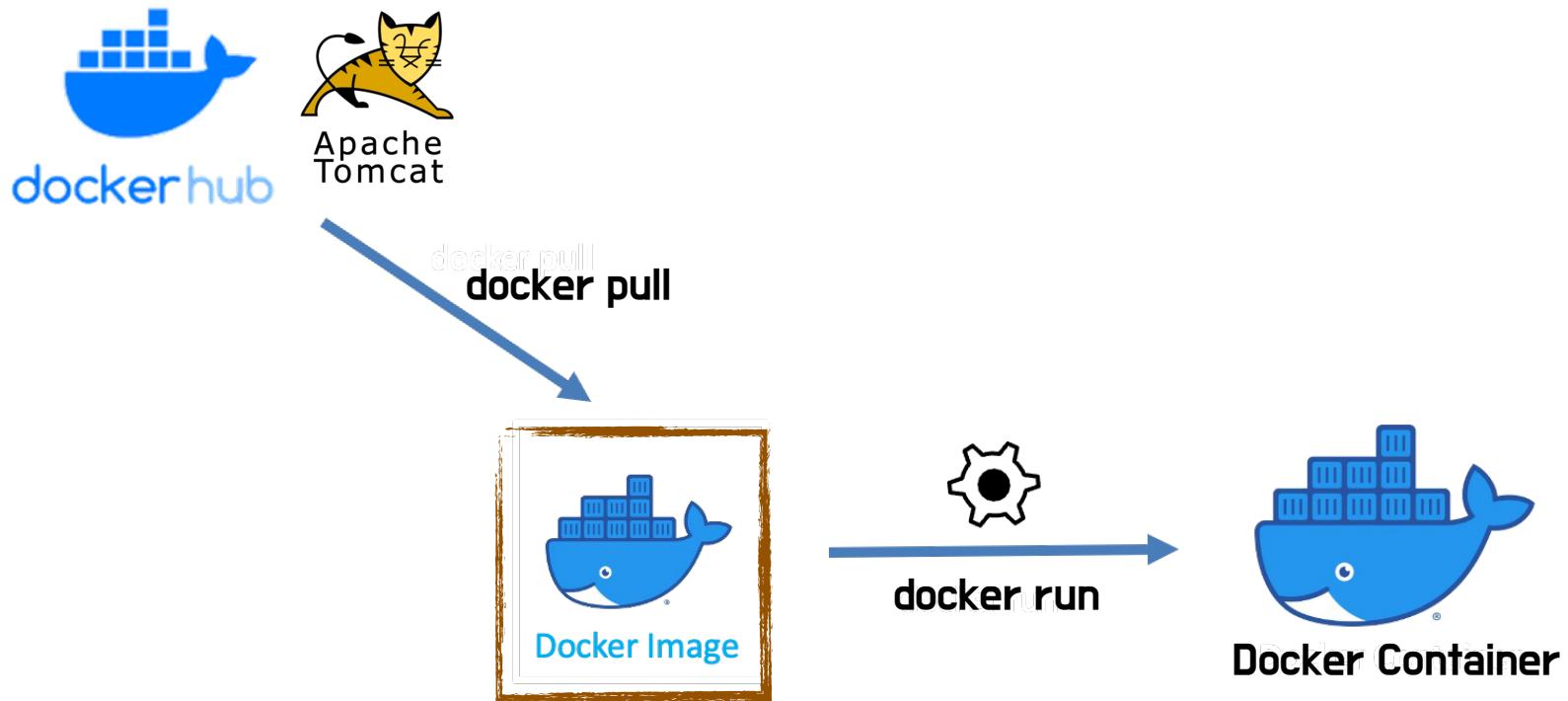
---

- Linux EC2 인스턴스 설정
- 도커 설치
- 도커 서비스 시작
- 도커 기본 명령어

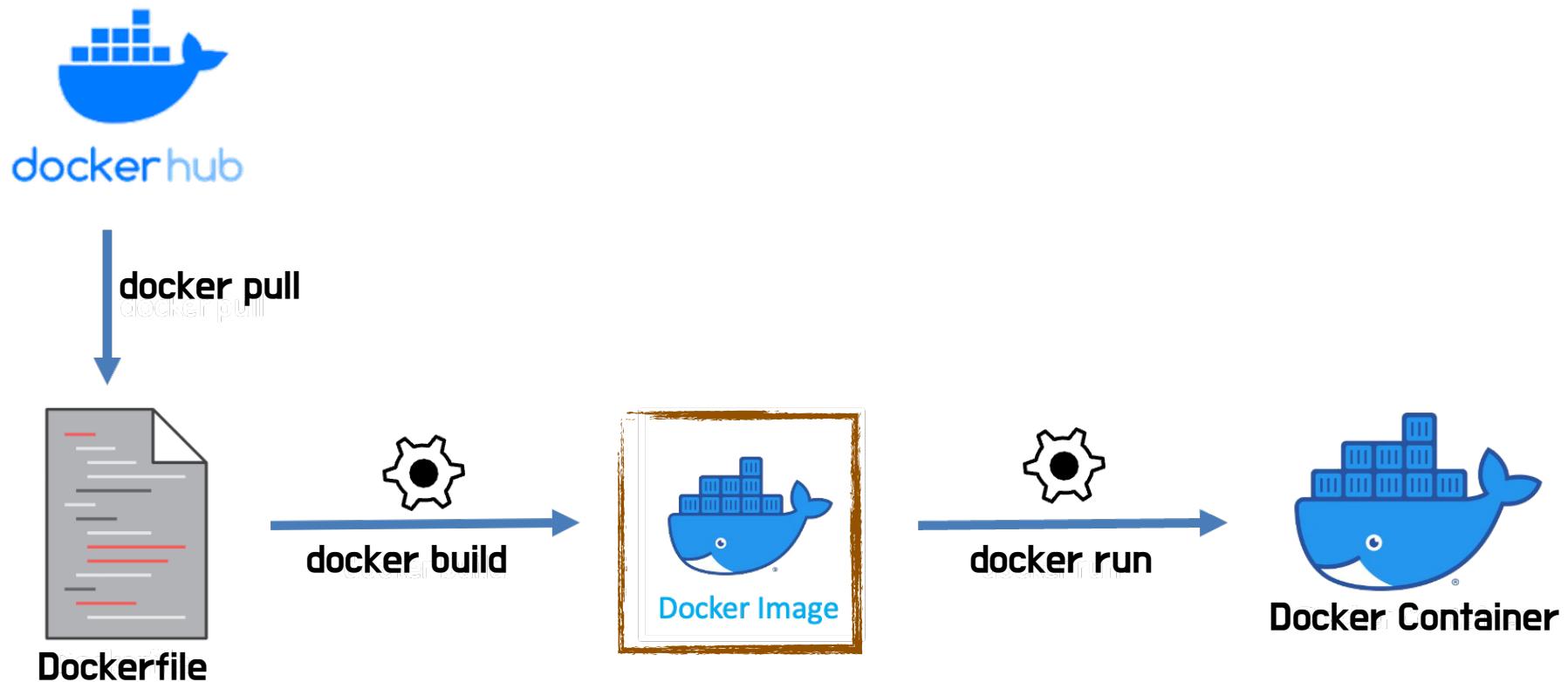


Amazon EC2

# 톰캣 도커 컨테이너 만들기



# 도커 컨테이너 만들기



# 첫번째 Dockerfile 만들기

---

- FROM: 베이스 이미지 pull
- RUN: 명령어 실행
- CMD: 컨테이너 실행 시 설정
- ENTRYPOINT: 실행 시작점 설정
- WORKDIR: working directory 설정
- COPY: 로컬 머신에서 도커 컨테이너로 디렉토리 복사
- ADD: 로컬 머신에서 도커 컨테이너로 파일과 폴더 복사
- EXPOSE: 실행시에 컨테이너의 리슨 포트 외부로 노출
- ENV: 환경변수 설정



# Centos 에 톰캣 설치 - Dockerfile

- Pull centos from dockerhub **FROM**
- Install java **RUN**
- Create /opt/tomcat directory **RUN**
- Change work directory to /opt/tomcat **WORKDIR**
- Download tomcat packages **ADD /RUN**
- Extract tar.gz file **RUN**
- Rename to tomcat directory **RUN**
- Tell to docker that it runs on port 8080 **EXPOSE**
- Start tomcat services **CMD**



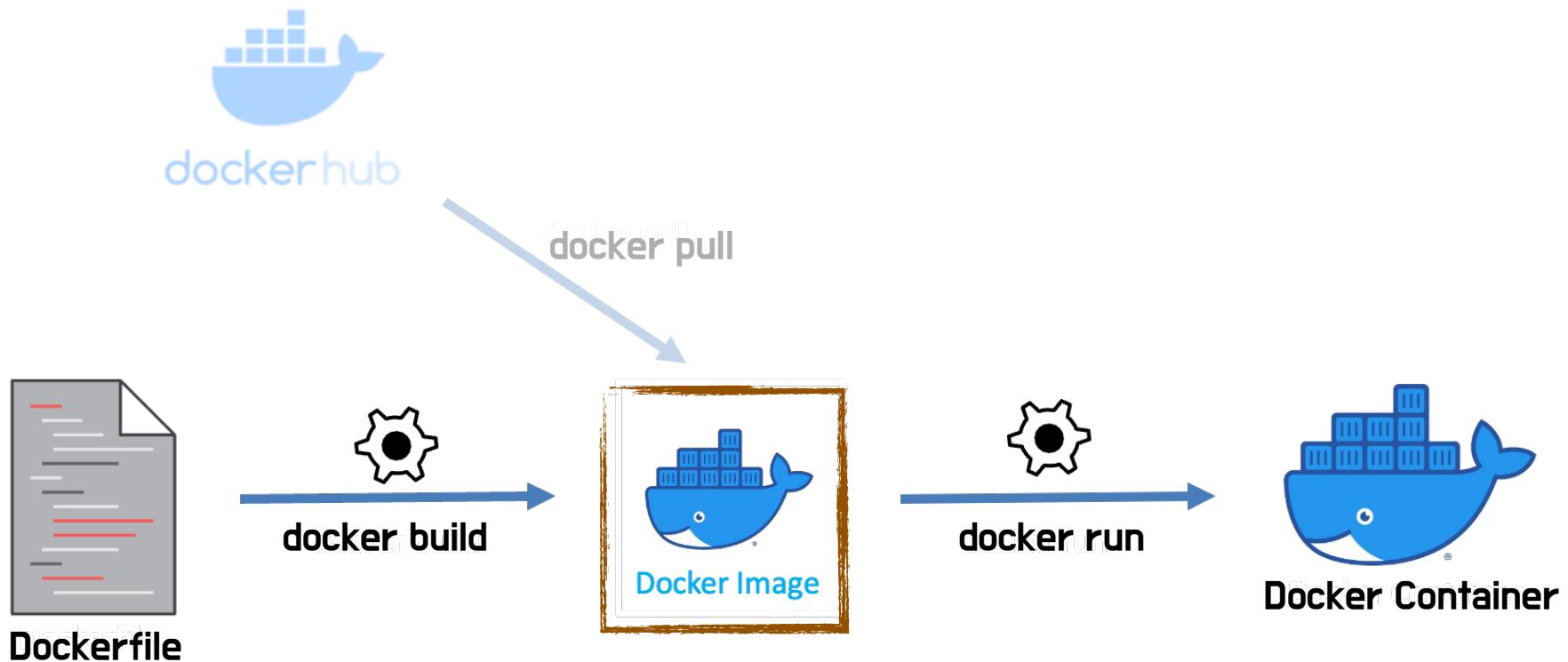
# Dockerfile

---

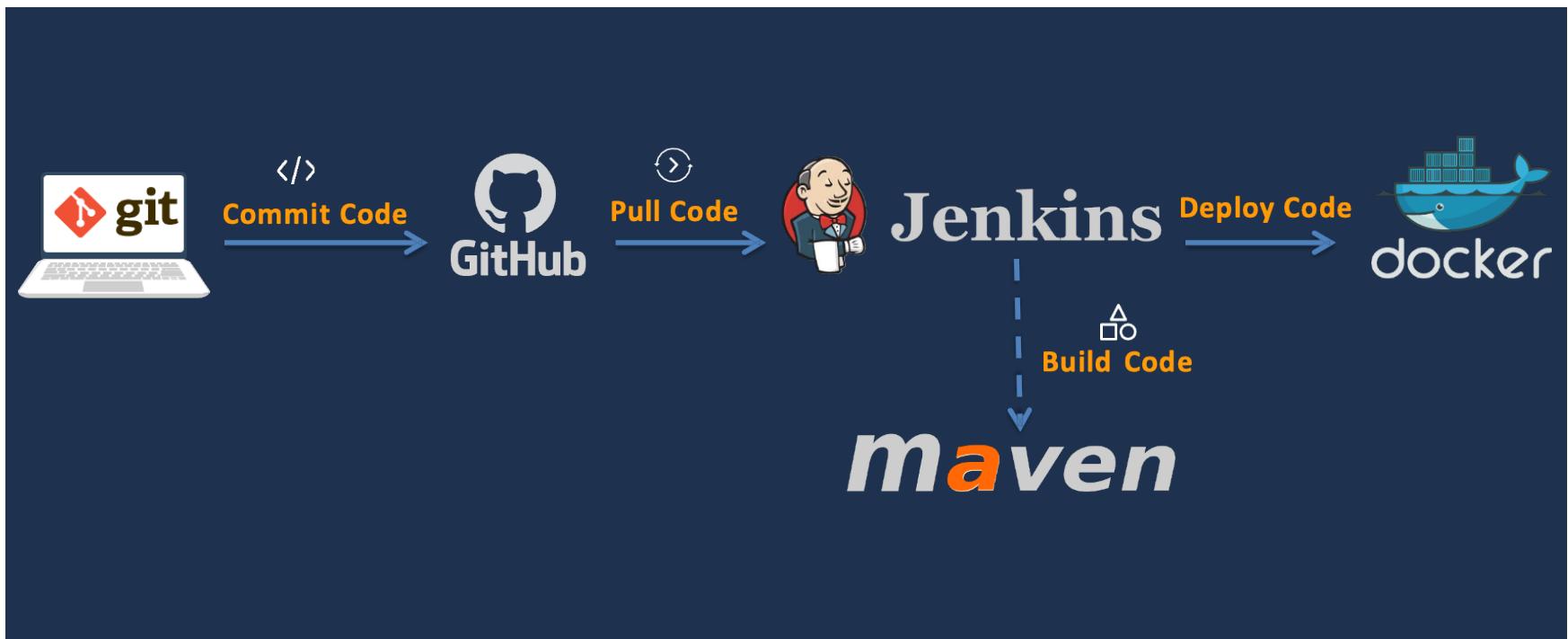
- FROM centos:centos7
- RUN yum -y install java
- RUN mkdir /opt/tomcat/
- WORKDIR /opt/tomcat
- ADD <https://dlcdn.apache.org/tomcat/tomcat-9/v9.0.54/bin/apache-tomcat-9.0.54.tar.gz> /opt/tomcat
- RUN tar xvfz apache\*.tar.gz
- RUN mv apache-tomcat-9.0.54/\* /opt/tomcat
- EXPOSE 8080
- CMD ["/opt/tomcat/bin/catalina.sh", "run"]



# 도커 컨테이너 만들기 (Dockerfile)



# 컨테이너에 배포



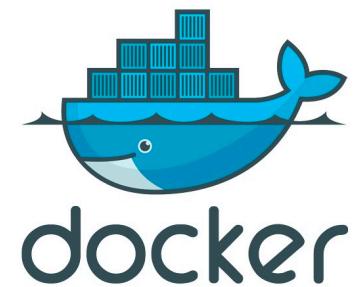
# jenkins에서 도커로 배포

---

- dockeradmin 유저 생성 (docker host)
- “Publish Over SSH” 플러그인 설치
- “configure systems” 메뉴에서 docker-host 추가
  - ssh를 통해 dockeradmin 계정으로 파일을 복사
  - dockeradmin home 디렉토리에 파일 복사



Jenkins



- Ansible 설치
- Ansible 인벤토리 생성
- 플레이북
- Ansible Vault

## 6. Ansible – IaaS 인프라 구성

# Ansible 이란?

---

- 구성 관리 도구, Configuration management tool
  - 수십, 수백대의 서버를 통합관리하고 구성하는데 적합
  - 이를 통해서 자동화 할수 있음
- 역등성, Idempotency
  - 여러 번 실행해도 항상 결과는 동일하다.
- agent-less
  - 대상 리소스에 에이전트 설치가 불필요
  - Chef, Puppet 은 agent 방식으로 동작 (장,단점이 있음)

# 엔서블 - RedHat

---

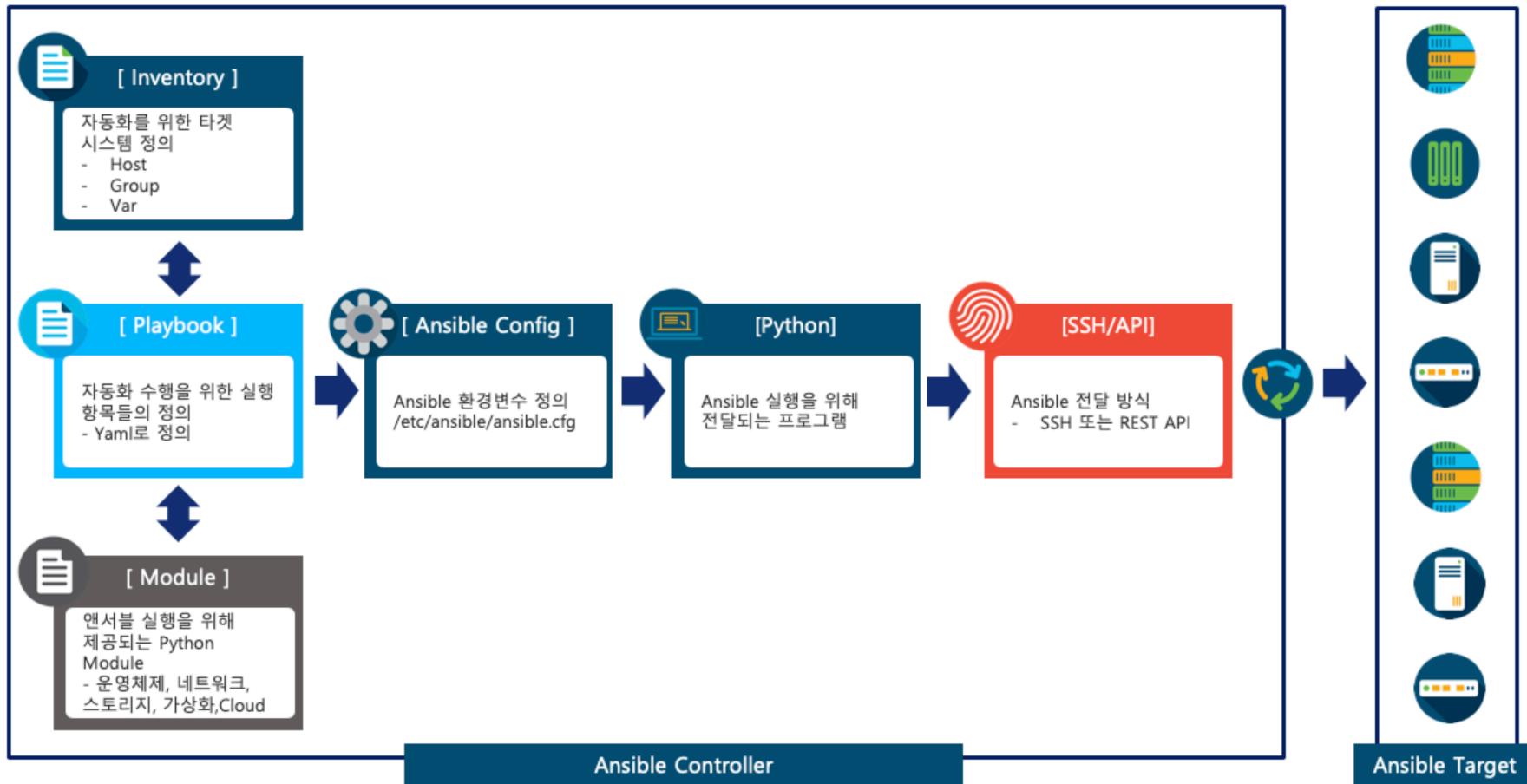
- Ansible 장점

- SSH 를 통한 구성으로 Agentless
  - ❖ 구성할 VM 에 에이전트를 설치할 필요가 없음
- YAML 사용으로 높은 접근성
- 다른 도구보다 훨씬 간소화
- 다양한 플랫폼 지원

- 단점

- 다른 도구에 비해 덜 강력함
- 변수 사용으로 인한 복잡성 증가

# Ansible 구조



# Ansible 구조

- Control Node

- Ansible이 설치된 Node

- Managed Node

- 관리대상 서버

- Inventory

- Managed Node 등록된 목록

- Modules

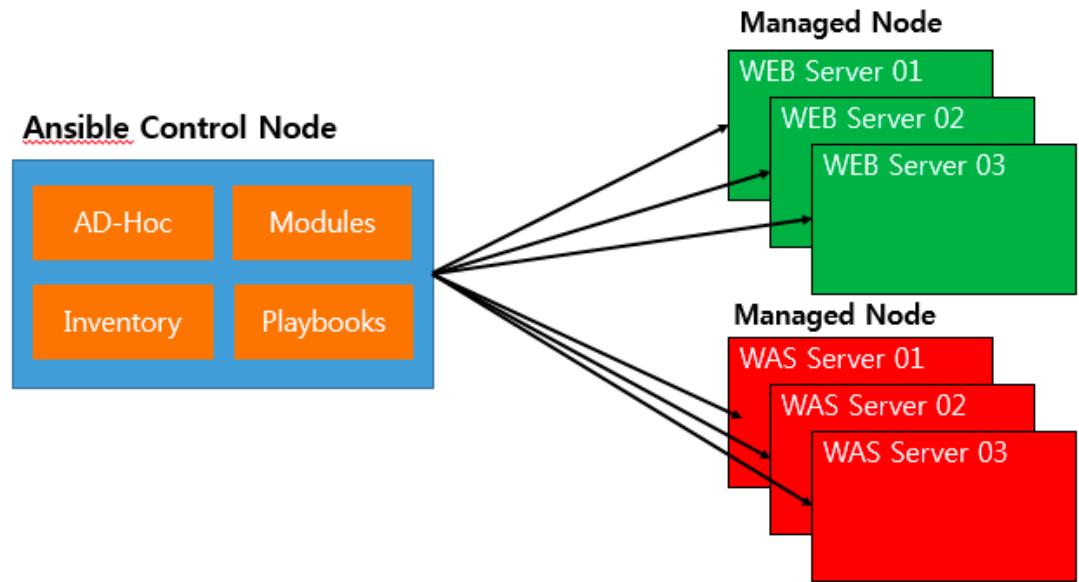
- Ansible에서 제공하는 실행단위 모듈

- Tasks

- Modules의 집합으로 작업단위

- Playbooks

- 계획된 작업을 순서대로 실행하기 위해 작성하는 YAML 파일



# YAML 파일 샘플

```
---
- name: get hostname
  hosts: localhost
  gather_facts: no

  tasks:
    - name: execute the hostname command
      command: hostname
      register: hostname_rs

    - debug:
        msg:
          - "{{ hostname_rs.stdout }}"

# YAML 파일은 --- 시작을 의미
# Playbook에 대한 이름 (호스트명을 가져오는 Playbook)
# 대상 호스트 (=localhost)
# facts에 대한 정보를 가져 올 것인지 여부

# tasks 시작을 의미
# 첫번째 tasks에 대한 이름 (hostname 명령어를 실행하라)
# command module을 사용하여 실행
# register에 실행 결과값을 저장

# 실행 결과 값을 확인하기 위한 debug
# 어떤 msg를 가져 올 것인지
# register에 저장된 내용에서 stdout을 보여줌
```

# 앤서블 설치

- ansible 이 설치되어 있지 않는 상태

```
[root@localhost ~]# ansible  
-bash: ansible: command not found
```

- Control Node 필수사항

- Python 2.6 또는 3.5 이상 설치되어야 하고, 윈도우 지원 않음

- Managed Node 필수사항

- ssh로 로그인 가능한 root 또는 sudo 권한이 있는 계정 필요
  - 네트워크 통신이 가능

- CentOS, RHEL 계열은 yum 으로 비교적 쉽게 설치 가능

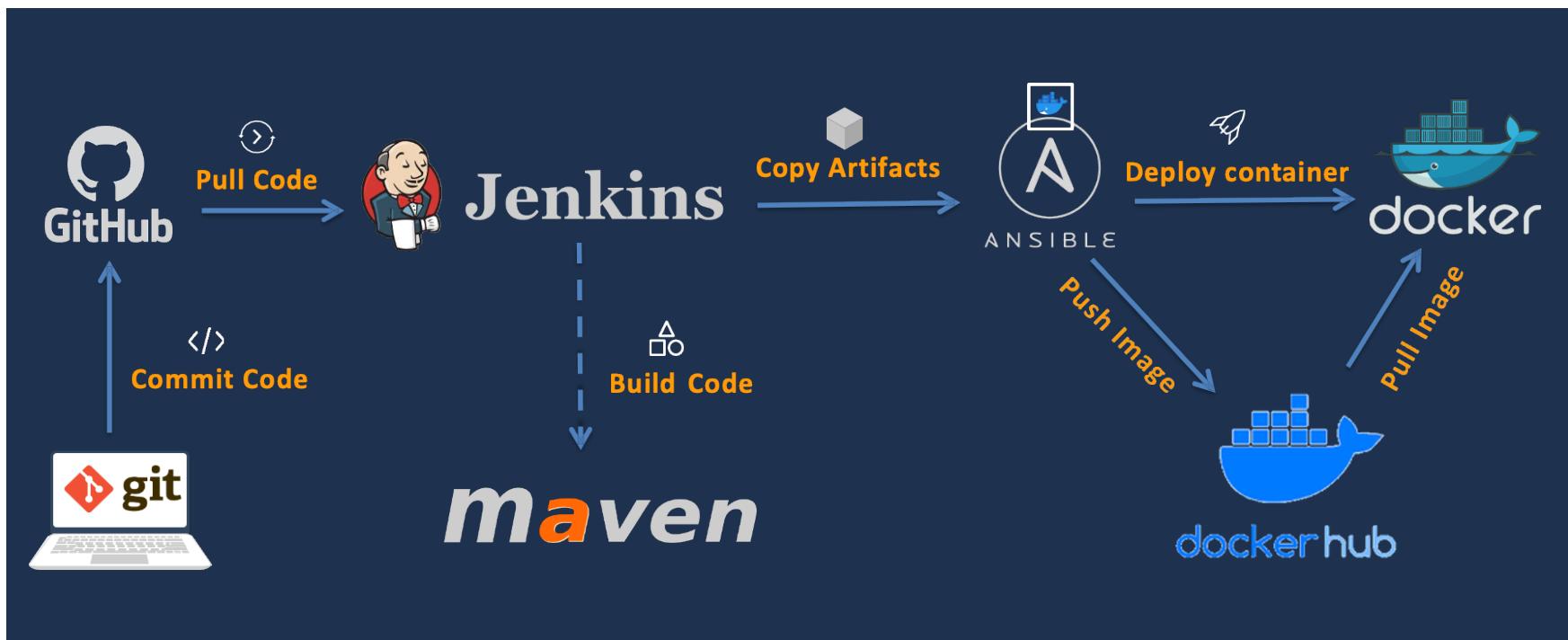
- 설치완료 후 버전 확인 - ansible --version

```
[root@localhost ansible-pkg]# ansible --version  
ansible 2.9.1  
  config file = /etc/ansible/ansible.cfg  
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']  
  ansible python module location = /usr/lib/python2.7/site-packages/ansible  
  executable location = /usr/bin/ansible  
  python version = 2.7.5 (default, Jun 20 2019, 20:27:34) [GCC 4.8.5 20150623 (Red Hat 4.8.5-36)]
```

- Ansible 설치
- 도커와 앤서블 연동
- Ansible Playbook

## 실습 3. CI / CD 파이프라인 Ansible 과 통합

# Ansible을 배포 툴로 사용



# Ansible 서버 준비

---

- EC2 인스턴스 설치
- hostname 세팅
- ansadmin 유저 생성
- sudoers 파일에 ansadmin 유저 추가
- ssh 키 생성
- 패스워드 기반 로그인 활성화
- ansible 설치



A N S I B L E

# Ansible 로 Docker 호스트 관리

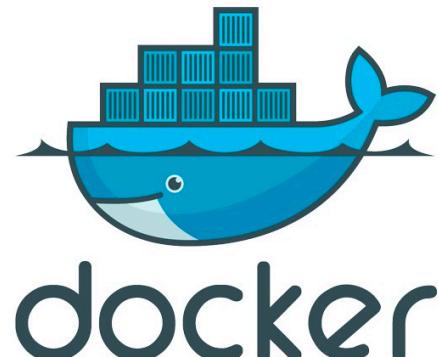
---

- Ansible 컨트롤 노드에 설정할 내용

- hosts 파일(인벤토리)에 추가
- ssh 키 복사
- 연결 테스트

- 도커호스트에 설정할 내용

- ansadmin 유저 생성
- ansadmin 유저 sudoers 파일에 추가
- 패스워드 기반 로그인 활성화



# jenkins와 연동

---



Jenkins

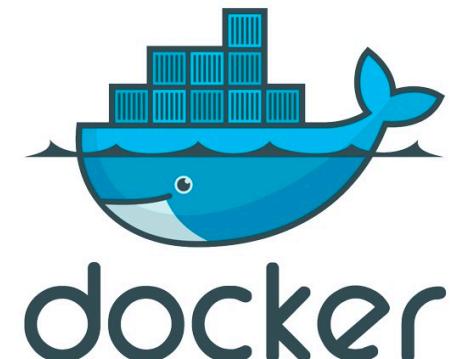


ANSIBLE

# Ansible playbook 배포

---

- 기존 컨테이너 제거
- 기존 이미지 제거
- 새로운 컨테이너 생성

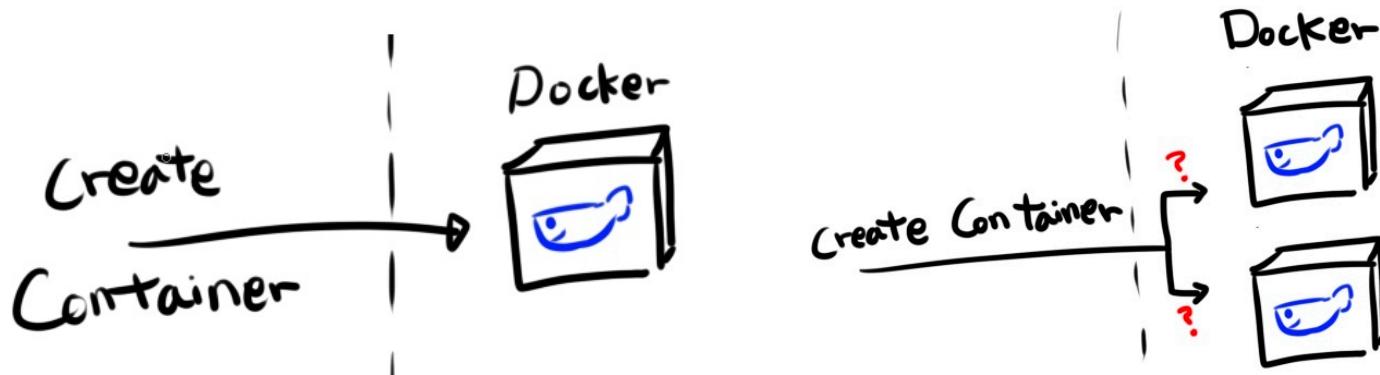


- 쿠버네티스 설치
- 쿠버네티스 애플리케이션 배포
- Helm 패키지 관리자
- AKS 사용
- Kubernetes 용 CI / CD 파이프라인 만들기
- 애플리케이션 매트릭 및 모니터링

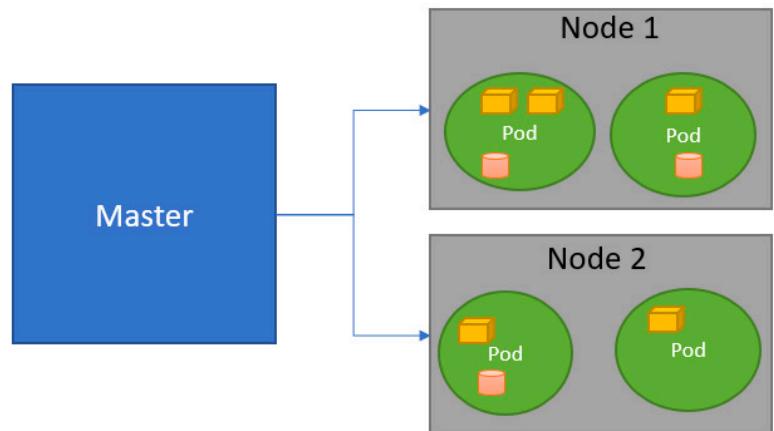


## 7. Kubernetes - 컨테이너 관리

# Kubernetes



- 여러 컨테이너를 관리하고 조정
  - Docker Swarm vs. K8s
  - 컨테이너 오케스트레이션
  - 단일 서버에서는 K8s 필요없음



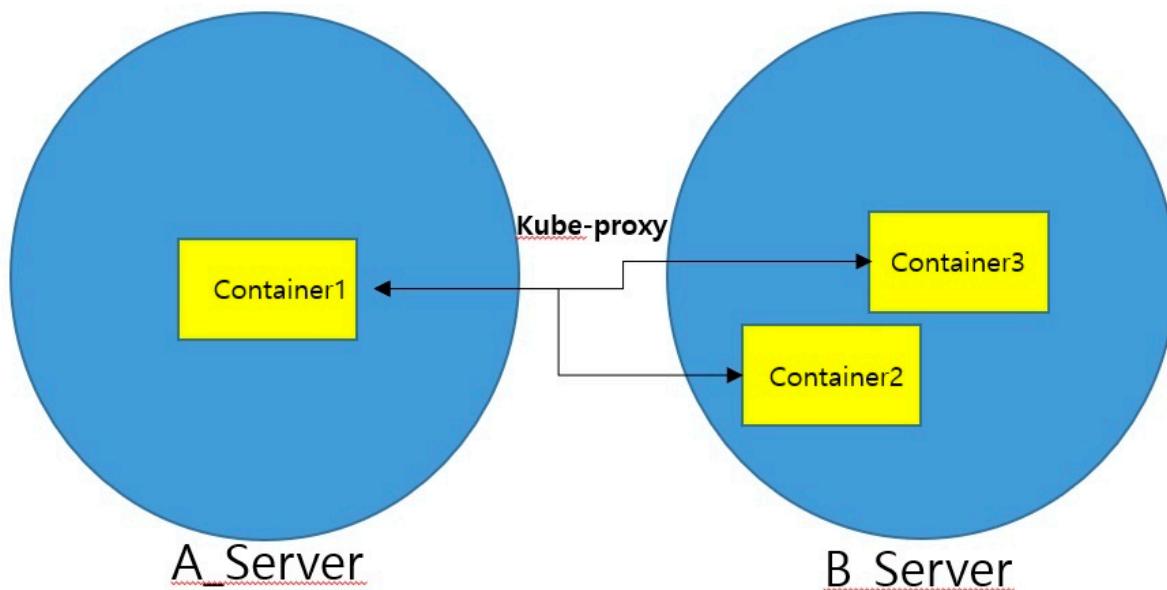
# Kubernetes 필요성

- 프로덕션 애플리케이션은 여러 컨테이너에 걸쳐 있음
  - 클러스터 전체에서 컨테이너 일정을 계획, 확장, 지속성 보장



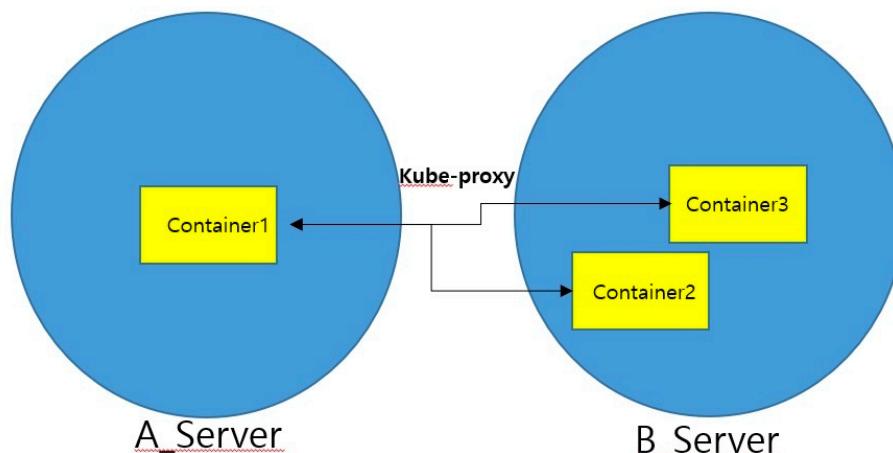
# kubernetes 목적

- 다중의 도커 서버를 하나의 Pool로 구성
  - 다중 서버의 도커 데몬에 연결하여 사용
  - 사용자는 도커 서버나 컨테이너가 몇개 실행중인지 알 필요가 없음
- 다중 서버에 분산되어 컨테이너 생성



# kubernetes 목적

- A서버 B서버 와의 컨테이너 통신
  - 각 서버 컨테이너 private ip 가 있음
  - A서버와 B서버 컨테이너 간 통신은 kube-proxy 등을 통해 통신
- 컨테이너 재 생성
  - 서버 다운 시 동일한 컨테이너 생성 - 서비스 지속성 유지
- Load Balance
  - Kubernetes 클러스트의 컨테이너 간 로드 밸런싱 수행 (round-robin 형식)

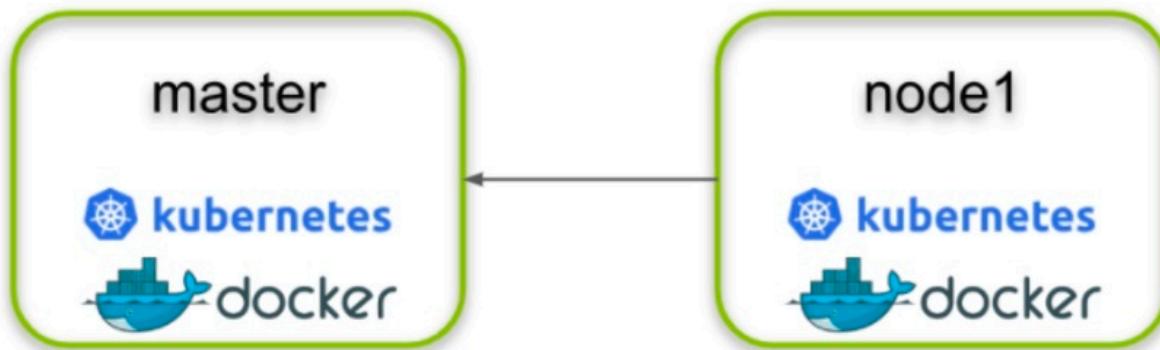


# Kubernetes 주요 용어

---

- master
  - 마스터 노드, 다중 도커 데몬을 관리
- worker
  - 도커가 설치되어 있으며, 실제 컨테이너들이 생성 일하는 노드, 마스터의 관리를 받고 있음
- pod
  - 쿠버네티스 기본 단위, 컨테이너 혹은 컨테이너의 묶음
- rc
  - replication controller, pod 을 자동으로 복제해주는 컨트롤러
- service
  - pod의 group을 식별하는 라벨이라는 기준에 따라 pod들을 하나의 서비스로 외부에서 접근 가능하게 해줌
- yaml
  - service, rc, pod 등 기능을 설명한 텍스트 형식의 코드

# Kubernetes 주요 도구



1) kubeadm init

2) kubeadm join

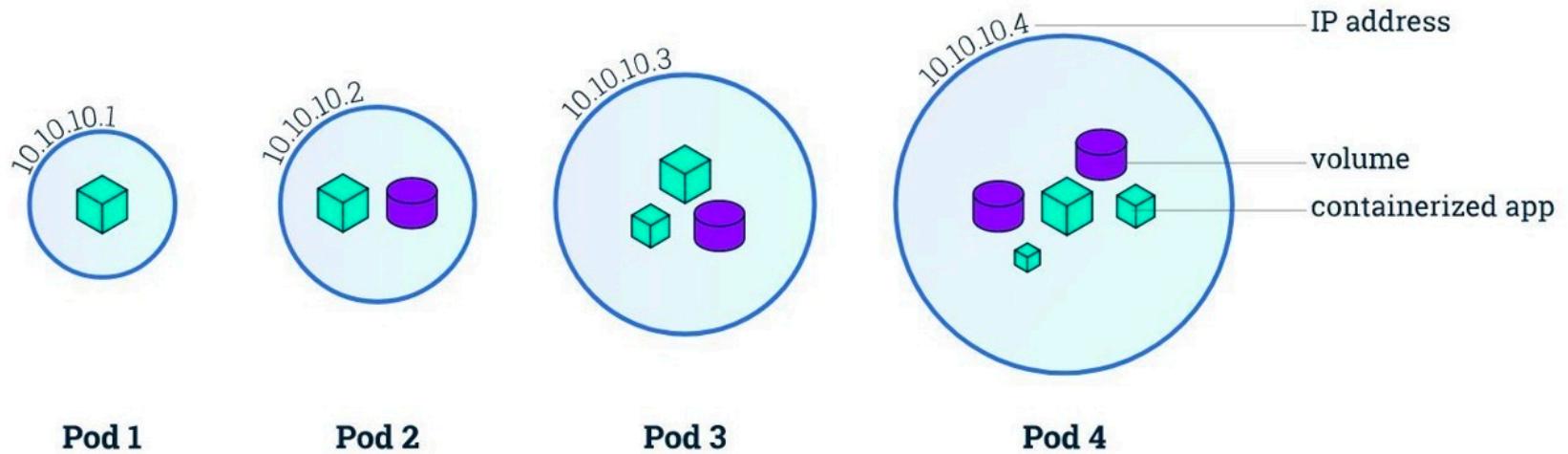
- **kubeadm :**

- init(join), 초기화(bootstrap)
- 쿠버네티스 구성과 초기화, 노드 확장 시 사용

- **kubectl :**

- cmd 작업수행, k8s object 생성, 관리

# Kubernetes pod



- 쿠버네티스의 기본 배포단위
  - 컨테이너를 포함한 단위
  - 컨테이너 단위로 배포하는 것이 아니라 pod 단위로 배포

# pod 생성 yaml

- **apiVersion**

- 주로 v1 사용

- **kind**

- 리소스 종류
- pod, service..

- **metadata**

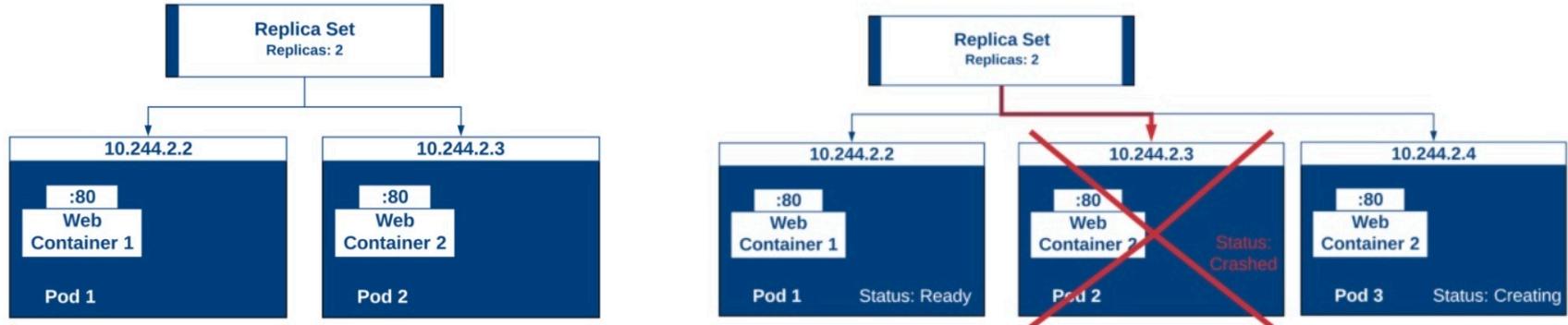
- 리소스의 메타데이터

- **spec**

- resource 등에 대한 상세정보

```
apiVersion: v1
kind: Pod
metadata:
  name: k8s-nodejs
  labels:
    app: hi-nodejs
spec:
  containers:
    - name: gpu-test
      image: 1985ck/gpu-test:1.0
    ports:
      - containerPort: 8000
```

# pod - replicaset



- web container 2개 복제로 정의하여 pod 생성
  - 컨테이너나 서버가 다운되면 다른 호스트에 컨테이너 생성
  - 쿠버네티스의 가장 강력한 기능 중 하나

# 쿠버네티스 설치 유형

개발 용도의 쿠버네티스 설치	Minikube Docker for Mac / Windows에 내장된 쿠버네티스
서비스 테스트 또는 운영 용도의 쿠버네티스 설치	kops kubespray kubeadm EKS, GKE, AKS 등의 관리형 서비스

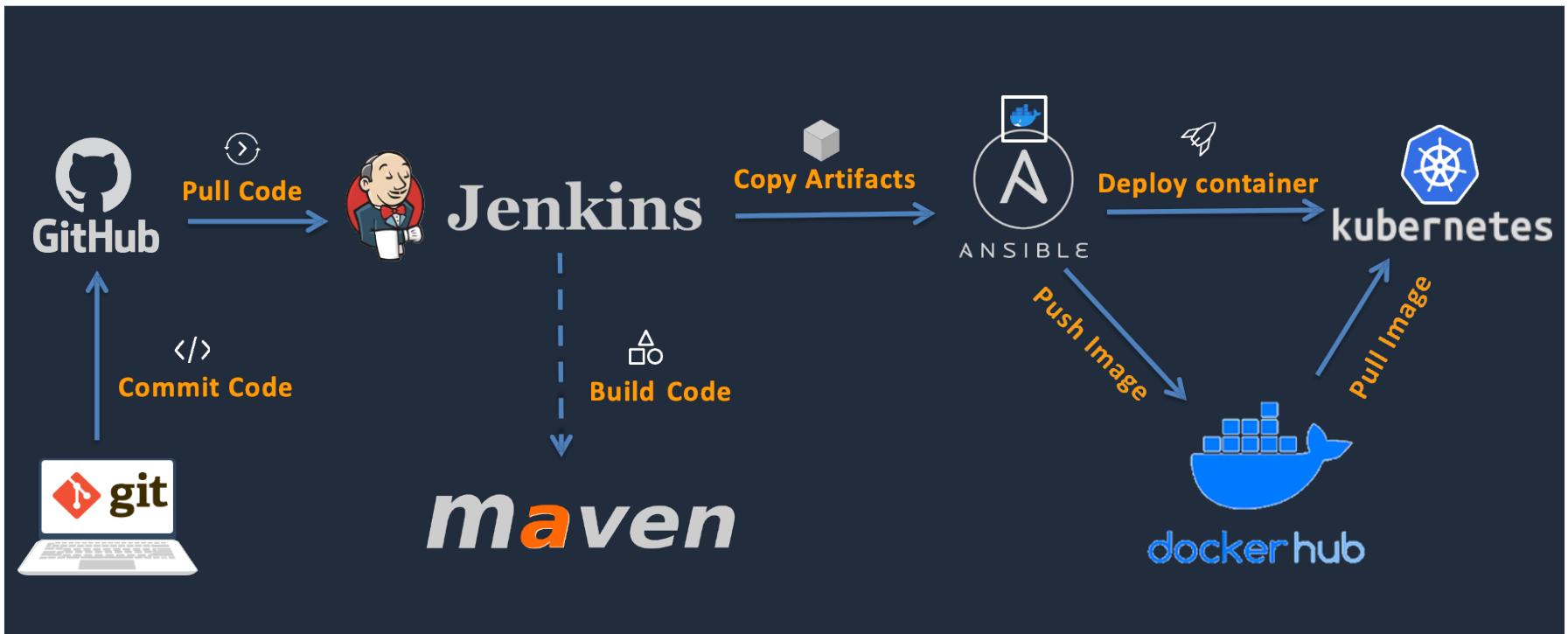
- EKS, GKE, AKS 등의 관리형 서비스 (Public Cloud)



- AWS EKS 사용
- Ansible 과 Kubernetes 연동

## 실습 4. Kubernetes에 배포하기

# Deploying on Kubernetes



# 실습 내용

---

- EKS에서 쿠버네티스 셋업 (eksctl 사용)
- kubectl 명령어로 데모 앱 디플로이
- 첫 번째 manifest 파일 작성
- 쿠버네티스와 Ansible 연동
- Ansible 플레이북으로 배포하기
- 젠킨스 사용



**kubernetes**

# Kubernetes 셋업

---

- 배포 툴

- kubeadm
- kops
- Kubespray



- Managed Services

- AWS EKS
- Azure AKS

# Kubernetes 셋업 방식

---

- kubeadm
- kops
- Kubespray

} 배포 툴



**kubernetes**

- AWS EKS
- Azure AKS

} 관리형 서비스

# EKS 셋업

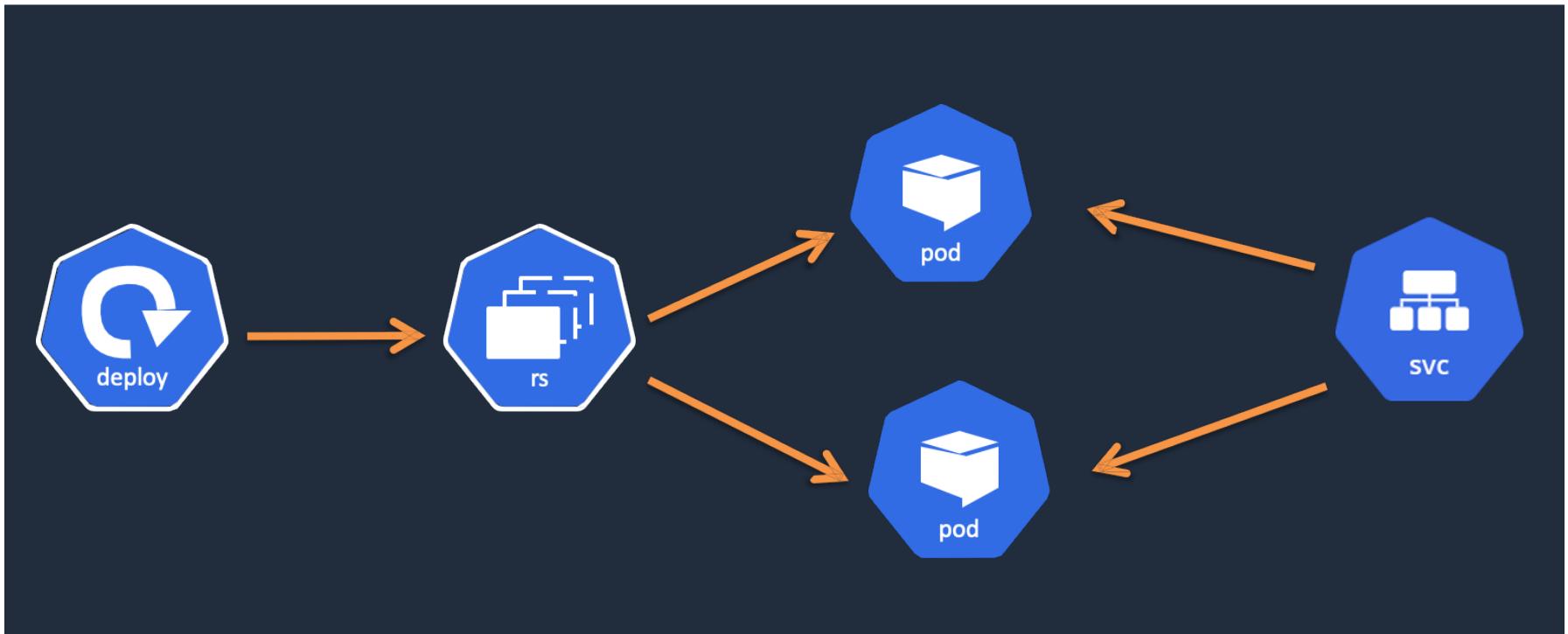
---

- EC2 인스턴스 생성 (부트스트랩 서버)
- AWSCLI 설치
- kubectl 설치 (/usr/local/bin 이동)
- eksctl 설치
- IAM 를 생성
- 클러스터 생성
- 클러스터 검증
- 클러스터 삭제



**kubernetes**

# Pod 생성



# 첫번째 Manifest 파일 작성

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: demo-app

spec:
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - name: nginx
      containerPort: 80
```

**pod.yml**

```
apiVersion: v1
kind: Service
metadata:
  name: demo-service

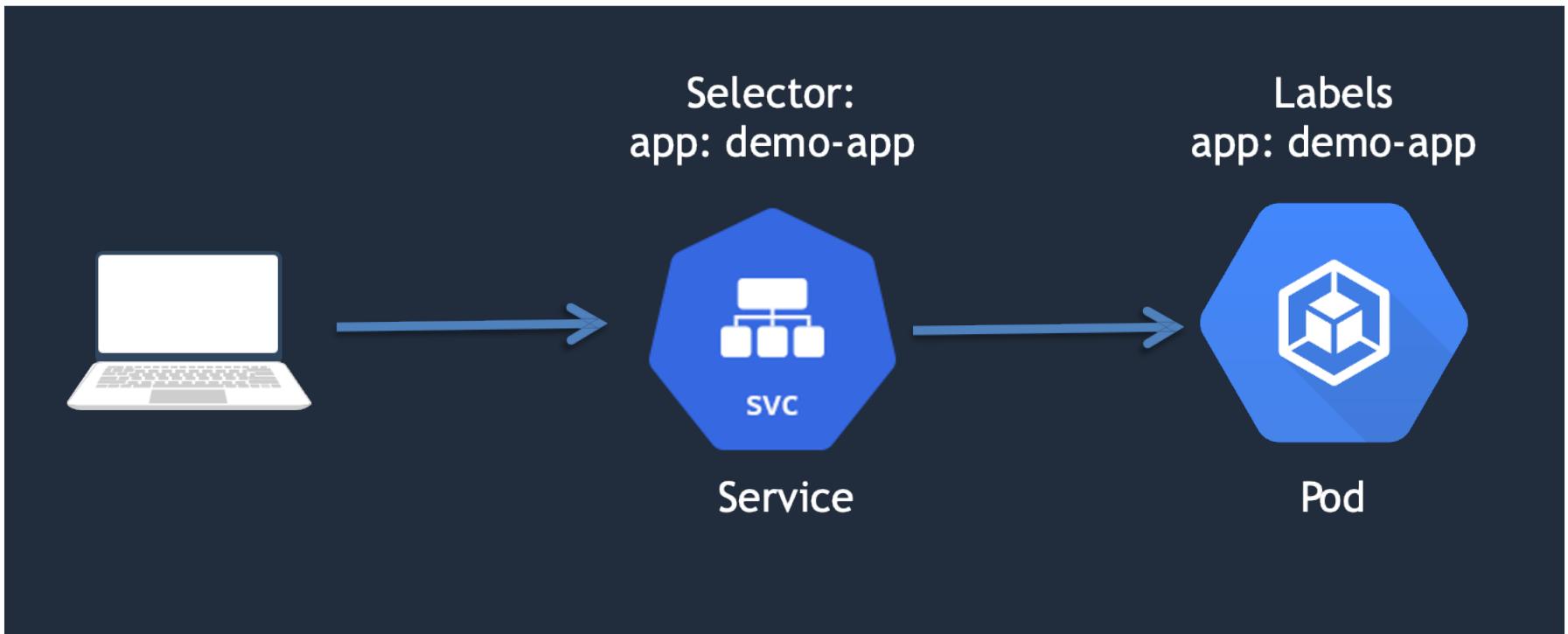
spec:
  ports:
  - name: demo-service
    port: 80
    targetPort: 80

  selector:
    app: demo-app
    type: LoadBalancer
```

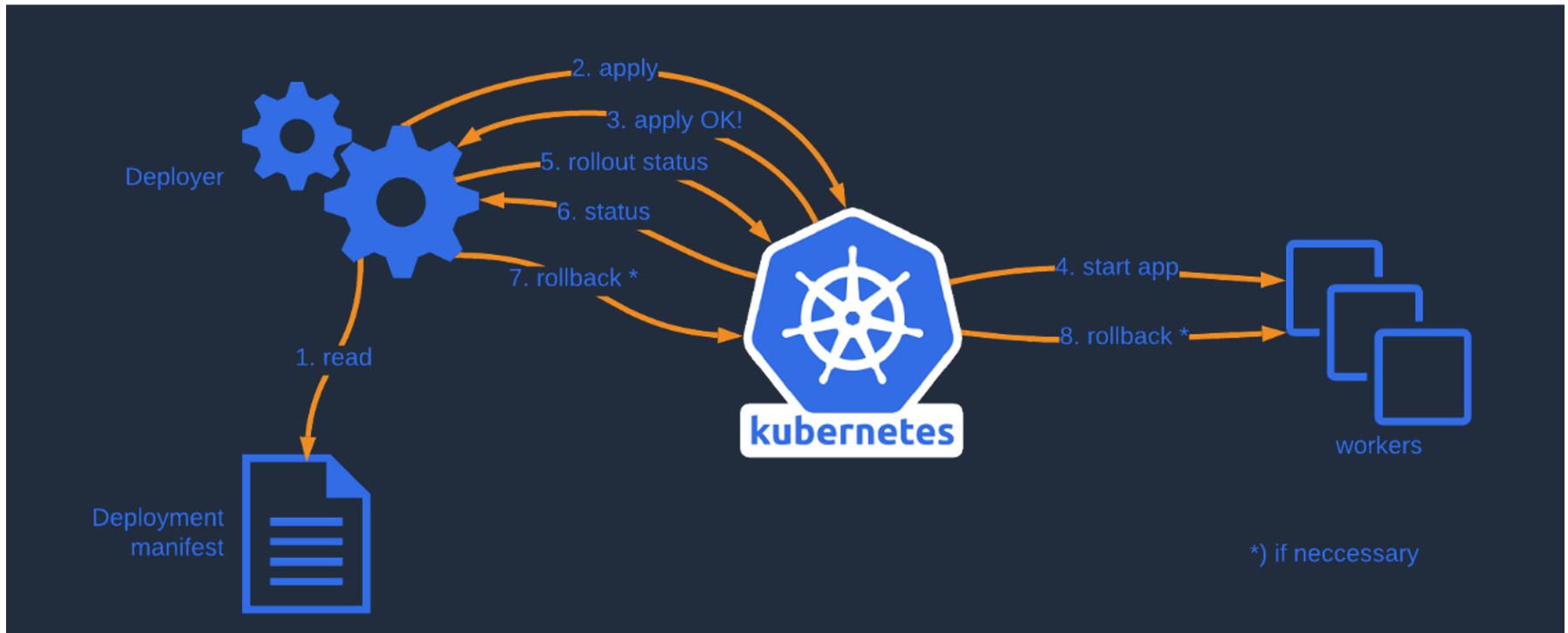
**service.yml**

**kubectl apply <filename>**

# 서비스와 Pod 셋업

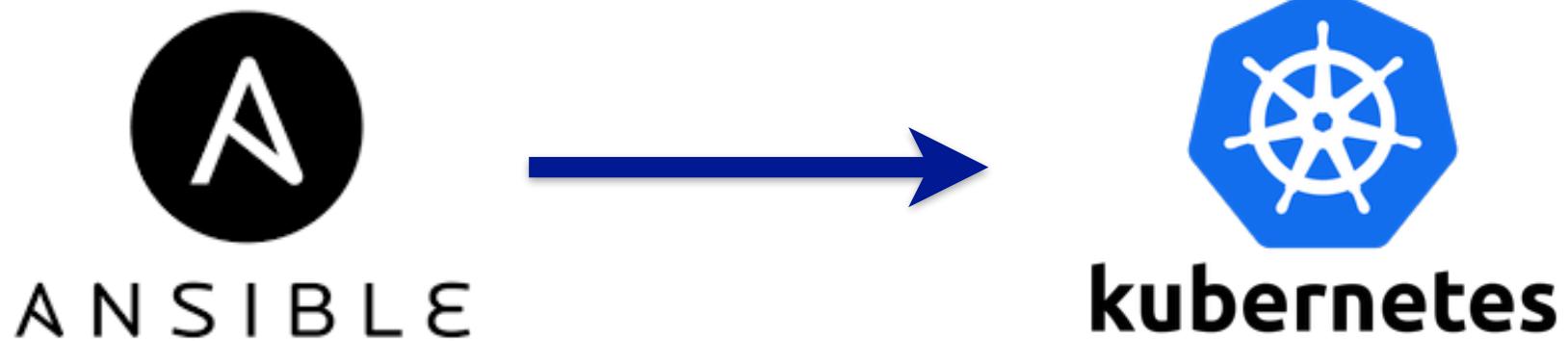


# 서비스와 Pod 셋업

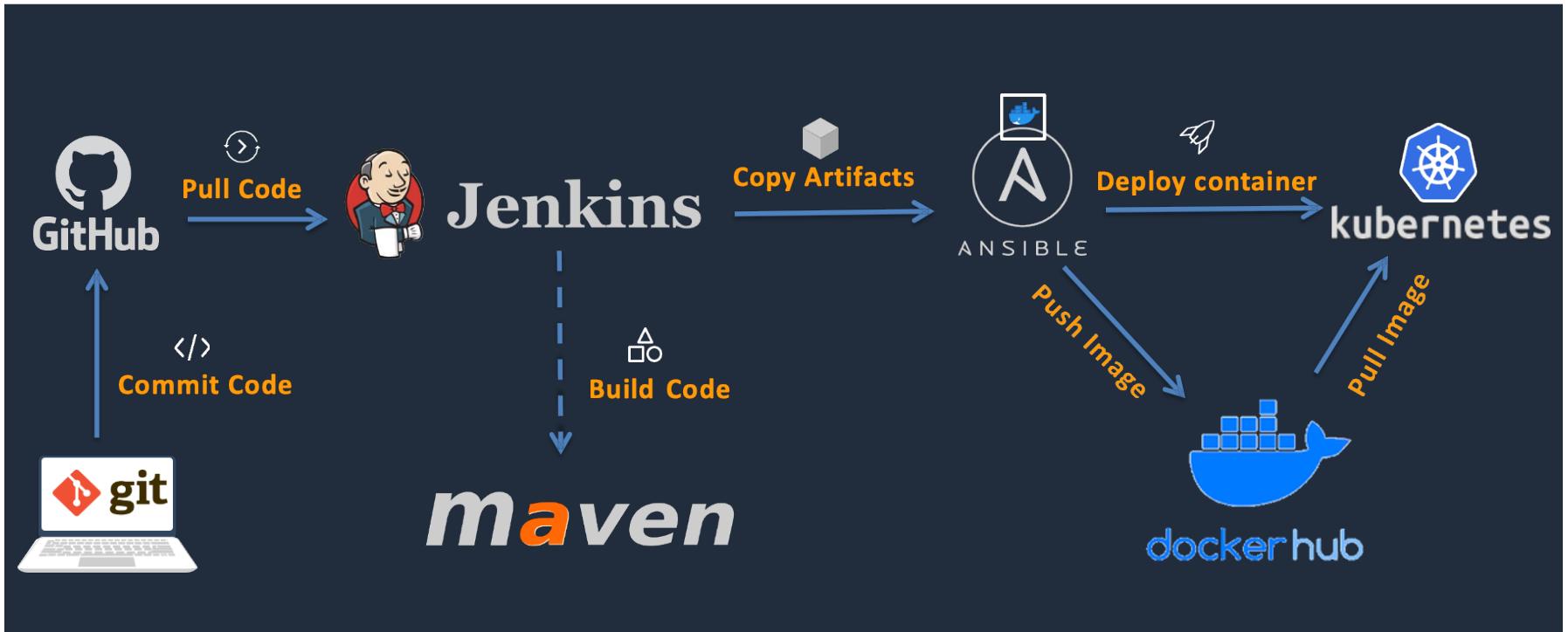


# 쿠버네티스와 Ansible 연동

---



# Deploying as a Pod



- 모든것을 자동화
- 올바른 도구 선택
- 모든 구성은 코드로 작성
- 시스템 아키텍처 설계
- 좋은 CI/CD 파이프라인 구축
- 테스트 통합
- DevOpsSec을 통한 보안 전환
- 시스템 모니터링
- 진화하는 프로젝트 관리

## 8. DevOps 모범 사례

# 과정 마무리

---

- DevOps 문화를 효과적으로 구현하기 위한 모범 사례

- IaC - 모든 것을 자동화 하라
- 올바른 도구를 선택하라
- 모든 구성은 코드로 작성하라
- 시스템 아키텍처 설계
- 테스트 통합
- 개발-보안-운영 (DevSecOps)로 보안을 포함
- 시스템 모니터링
- 진화하는 프로젝트 관리

# 모든 것을 자동화 하라

---

- DevOps 문화의 목적
  - 애플리케이션 릴리즈를 더 짧은 주기로 더 빠르게 제공
- 모든 작업을 자동화
  - 애플리케이션과 해당 인프라 배포
  - 테스트 자동화
  - 보안 관련 설정 자동화
  - 스크립트에서 자동화 하여 CI / CD 파이프라인에 통합
  - 배포 모니터링 강화 -> 문제 발생 시 빠르게 백업 및 복원
  - 자동화 및 오케스트레이션 도구 적용 및 구현

# 올바른 도구를 선택하라

---

- 좋은 도구를 어떻게 선택하나?

- DevOps 문화 -> Dev, Ops, Process, 도구의 결합
- Dev 와 Ops 가 공유하고 사용할 수 있어야 하며
- 동일한 프로세스에 통합 되어야 함
- 오픈소스와 유료 툴(AWS, Azure ..)

# 모든 구성을 코드로 작성

---

- **IaC**

- Terraform, Ansible, Packer
- JavaScript, JSON, Bash, PowerShell, Python 스크립트
- 운영팀과 개발팀 모두를 위한 프로젝트 초기부터 고려되어야 함

# 시스템 아키텍처 설계

---

- **소프트웨어 엔지니어링 진화**
  - 폭포수 방식 -> 매자일 방식 -> DevOps 문화
  - 진화는 애플리케이션 뿐 아니라 인프라에서도 많은 변화와 개선을 가져 옴
- **클라우드 네이티브**
  - 분석 단계 부터 클라우드 또는 DevOps 문화에 적합하게 설계를 고려 해야 함
    - ❖ 클라우드 아키텍트, 개발자(솔루션 아키텍트), 보안팀의 협력
- **マイクロ 서비스**

# 좋은 CI / CD 파이프라인 구축

---

- CI / CD 파이프라인 구축은 DevOps에서 필수
  - 프로젝트 시작 초기 단계에 설정
  - CI 에서의 팀 구성원에게 빠른 피드백 제공
  - 통합테스트 같은 오래 걸리는 작업은 밤에 예약으로 수행
  - 민감한 정보(암호, 연결 문자열, 토큰..)는 권한 제어
    - ❖ 중앙집중식 비밀 관리도구 - Vault

# 테스트 통합

---

## ● 테스트 자동화

- DevOps에서의 또 하나의 핵심
- 최소한 애플리케이션 단위테스트 실행을 통합
  - ❖ TDD 등으로 CI 파이프라인에 통합
- 통합테스트는 애플리케이션 작동 품질보장 테스트
  - ❖ 시간이 오래 걸리는 작업일 때는 밤에 예약으로 실행
- 테스트 실행이 실패할 경우 전체 프로세스가 중단
  - ❖ 해당 기능 비활성화의 득과 실을 잘 따져야 함

# DevSecOps 를 통한 보안 전환

---

- 보안 및 규정 준수 분석도 DevOps 프로세스의 일부
  - 보안 규칙에 대한 개발 팀 간 인식이 부족
  - 애플리케이션 코드 보안
  - DevOps 와 보안 사이의 장벽을 제거

# 시스템 모니터링

---

- 지속적으로 모니터링 할 도구의 구현
  - 모든 팀과 시스템의 모든 수준에서 구현되어야 함
- 모니터링 대상
  - 애플리케이션 사용에 대한 정보를 수집
    - ❖ 로깅 또는 추적 시스템 (ELK, Splunk..)
  - 인프라 상태를 측정하고 모니터링
    - ❖ VM, RAM, CPU, 네트워크 대역폭
  - DevOps 프로세스 상태
    - ❖ 파이프라인 수행시간, 파이프라인 요소 개수 등. 실행에 대한 메트릭
- Prometheus, Grafana, New Relic, Nagios ..

# 진화하는 프로젝트 관리

---

- 변화를 통해서만 구현되고 실현될 수 있음
  - 애플리케이션의 짧은 배포 주기
    - ❖ 애자일 방법론, 스프린트 (2~3주의 짧은 주기)
  - 조직 구성의 변화
    - ❖ 전문분야별로 팀을 구성 -> 여러 분야의 전문가가 팀을 이룸
    - ❖ 팀에는 개발자, 운영, 테스트가 있으며, 동일한 목표를 가짐

# 수고하셨습니다.

김순곤

[soongon@hucloud.co.kr](mailto:soongon@hucloud.co.kr)