2025 DevSecOps 환경 구축 가이드

상징, 로고, 폰트, 원이(가) 표시된 사진

자동 생성된 설명

Korea IT DevSecOps

2025. 02. 20

# 서론

**2025 DevSecOps 환경 구축 가이드**는 DevSecOps 방식을 통해 개발, 운영, 보안을 효과적으로 통합하여 CI/CD 프로세스를 최적화하고 시스템의 안전성을 강화하는 방법을 제시합니다. 특히, Jenkins, SonarQube, Docker, Kubernetes와 ELK Stack(Elasticsearch, Logstash, Kibana)을 활용하여 소스 코드 품질을 개선하고 보안 자동화를 구현하는 구체적인 접근법을 제공합니다.

이 문서는 다음을 목표로 합니다:

1. 개발 산성 증대

CI/CD 파이프라인 자동화를 통해 효율적인 빌드 및 배포 프로세스 구현.

2. 보안 내재화

코드 분석 및 취약점 점검 도구를 활용하여 개발 초기 단계에서부터 보안을 강화.

3. 안정적인 환경 구축

Kubernetes 기반 확장 가능하고 안정적인 애플리케이션 실행 환경 설계.

4. 실시간 모니터링 및 분석

ELK Stack을 사용하여 시스템 로그와 보안 이벤트를 통합 관리하고, 빠른 문제 분석과 대응지원.

5. 협업화 촉진

개발, 운영, 보안 팀 간의 긴밀한 협력을 통해 조직 전반의 보안 인식 제고.

본 가이드는 실무적인 설정 및 적용 사례를 중심으로 DevSecOps 환경을 단계적으로 구축하는 방법을 설명하며, 이를 통해 안전하고 효율적인 소프트웨어 개발, 운영, 그리고 보안을 제공합니다.

# 팀 소개

|  |  |
| --- | --- |
| 상징, 로고, 폰트, 원이(가) 표시된 사진  자동 생성된 설명  Team DevSecOps | 기획  정재호 장진영 박진우 안정훈 지윤정 |
| 개발  배준성 윤광혁 최민재 장태경 |
| 운영  김범준 김지홍 노윤서 손혜영 이주원 |

# Lab Git Repository

<https://github.com/jinyeong001/DevSecOps.Full-Project.git>

<https://github.com/GH6679/web_wargamer.git>

목차

[1. DevSecOps 5](#_Toc186704721)

[1.1. DevSecOps 란? 5](#_Toc186704722)

[1.2. DevSecOps의 이점 6](#_Toc186704723)

[2. 사전 준비 사항 7](#_Toc186704724)

[2.1 사전 요구 사항 (필요 설치 파일) 7](#_Toc186704725)

[2.2 사전 요구 사항 설치 7](#_Toc186704726)

[3. yaml 작성법 9](#_Toc186704727)

[3.1 yaml 이란? 9](#_Toc186704728)

[3.2 주요 필드 9](#_Toc186704729)

[3.3 워커 노드 추가 10](#_Toc186704730)

[3.4 기타 주요 필드 11](#_Toc186704731)

[4. Kubernates 14](#_Toc186704732)

[4.1 kind 14](#_Toc186704733)

[4.2 오류 발생 또는 충돌 시 16](#_Toc186704734)

[5. Jenkins 17](#_Toc186704735)

[5.1 Jenkins 란? 17](#_Toc186704736)

[5.2 DevSecOps에서의 Jenkins의 역할 17](#_Toc186704737)

[5.3 Jenkins worker node 생성 18](#_Toc186704738)

[5.4 Jenkins worker node 실행 21](#_Toc186704739)

[6. ELK 23](#_Toc186704740)

[6.1 ELK 란? 23](#_Toc186704741)

[6.2 ELK worker node 생성 23](#_Toc186704742)

[7. Web 구축 28](#_Toc186704743)

[7.1. WebPage 소스코드 28](#_Toc186704744)

[7.2. Web worker node 생성 29](#_Toc186704745)

[7.3 Web worker node 실행 33](#_Toc186704746)

# 1. DevSecOps

### 1.1. DevSecOps 란?

DevSecOps는 개발(Development), 보안(Security), 운영(Operations)을 통합하여 소프트웨어 개발의 전 과정에서 보안을 자동화하고 내재화하는 방법론입니다. 기존 DevOps 모델은 개발과 운영 간 협업에 초점을 맞췄으나, 보안은 종종 후순위로 다뤄졌습니다. 이에 반해 DevSecOps는 초기 설계 단계부터 배포, 운영에 이르기까지 보안을 모든 과정에 포함시키며, 이를 통해 **보안은 모두의 책임**이라는문화를 강조합니다.

s

DevSecOps의 주요 원칙은 다음과 같습니다:

* **자동화된 보안 통합**: CI/CD 파이프라인에 보안 도구를 통합하여 코드 분석, 취약점 점검, 종속성 관리 등을 자동화합니다.
* **보안 중심의 협업**: 개발, 운영, 보안 팀이 함께 작업하며 보안 문제를 조기에 발견하고 해결합니다.
* **지속적인 개선**: 위협 모델링과 실시간 모니터링을 통해 보안을 지속적으로 강화합니다.

우리는 DevSecOps 구현을 위해 Jenkins와 같은 CI/CD 도구를 기반으로 SonarQube로 코드 품질 및 보안 점검을 수행하며, ELK Stack을 활용해 시스템 로그를 분석하고 이상 행동을 탐지합니다. 또한, Docker와 Kubernetes를 통해 컨테이너 보안을 강화하고, Falco 등의 런타임 보안 도구를 사용하여 운영 환경의 보안성을 유지합니다.

이러한 접근법은 보안을 단순한 방어 전략이 아닌 개발과 운영의 핵심 요소로 자리 잡게 합니다. 이를 통해 조직 내 보안 문화가 정착되고, 빠르고 안전한 소프트웨어 배포가 가능해집니다.

### 1.2. DevSecOps의 이점

DevSecOps를 채택하면 다양한 이점을 누릴 수 있습니다. 먼저, CI/CD 파이프라인에서 자동화된 보안 테스트와 취약점 분석 도구를 활용함으로써 개발 초기부터 보안을 강화할 수 있습니다. 보안이 개발 초기 단계에 통합되면, 후반 단계에서 보안 취약점을 수정하는 데 드는 비용을 크게 절감할 수 있습니다. 이는 보안 문제를 사전에 식별하고 해결하는 데 중요한 역할을 하며, 전반적인 개발 비용 절감으로 이어집니다.

또한, DevOps의 자동화와 속도를 유지하면서도 보안 검사를 주기적으로 수행할 수 있습니다. 이를 통해 안전한 코드 배포가 보장되며, 지속적인 통합 및 배포(CI/CD)를 지원하는 보안 관행이 자연스럽게 형성됩니다. 주기적인 보안 점검을 통해 개발과 배포 과정에서 발생할 수 있는 보안 리스크를 사전에 최소화할 수 있습니다.

DevSecOps에서는 보안이 단지 보안 팀의 책임만이 아닙니다. 개발자와 운영 팀도 보안에 대한 책임을 공유하고 협업하는 문화가 필요합니다. 이를 통해 조직 내 보안 인식 수준이 높아지고, 더 빠르고 효율적인 보안 대응이 가능합니다. 또한, 위협 모델링과 모니터링을 통해 실시간으로 보안을 관리하고, 발생할 수 있는 보안 문제를 예방할 수 있습니다. 이러한 방식은 다양한 산업에서 요구하는 보안 표준과 규제를 준수하는 데도 유효합니다.

이와 같은 이유로 우리는 DevSecOps를 채택하여 개발, 보안, 운영이 유기적으로 협력하고 효율적으로 관리될 수 있도록 하는 통합 프로젝트를 추진하고 있습니다. DevSecOps 환경을 통해 보안을 강화하면서도 개발과 운영의 효율성을 유지할 수 있는 시스템을 구축할 수 있습니다.

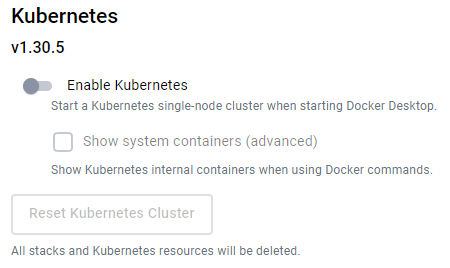
# 2. 사전 준비 사항

### 2.1 사전 요구 사항 (필요 설치 파일)

* WSL(WSL2)
* Docker Desktop(Kind)
* Git
* Chocolatey
* Kuberctl
* 구축파일(.yaml)

본 프로젝트의 작업은 바탕화면에서 진행됩니다.

Docker Desktop 설치 후 설정에서 Kubernetes 해제 필요 (기본 설정)



### 2.2 사전 요구 사항 설치

Window PowerShell 관리자 권한으로 실행

1) WSL(Windows Subsystem for Linux) 설치

Windows에서 Linux 환경을 구축할 수 있게 해주는 핵심 기능을 한다.

# WSL 설치

1. dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
2. dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

# WSL2 기본 버전으로 설정

1. wsl –set-default-version 2

2) Chocolatey 설치

Chocolatey는 Windows용 패키지 관리자 도구이다.

# Chocolatey 패키지 관리자 설치

1. Set-ExecutionPolicy Bypass -Scope Process -Force
2. [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072
3. iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

3) Git 설치

Github 및 Jenkins를 통한 자동화 핵심 기능을 한다.

# Git 기본 패키지 설치

1. choco install git -y

4) Docker Desktop, kubectl과 kind 설치

Kind(Kubernetes IN Docker)는 Docker 컨테이너 노드를 사용하여 로컬 Kubernetes 클러스터를 실행하기 위한 도구이다.

# Docker Desktop 설치

1. choco install docker-desktop -y

# kubectl과 kind 설치

1. choco install kubernetes-cli -y
2. choco install kind -y

# 3. yaml 작성법

### 3.1 yaml 이란?

YAML은 "YAML Ain't Markup Language"의 약자로, 데이터 직렬화에 사용되는 가볍고 사람이 읽기 쉬운 형식입니다. JSON이나 XML과 비교해 간결하고 이해하기 쉬운 구문을 제공하며, 설정 파일이나 데이터 저장에 자주 활용됩니다. YAML은 들여쓰기를 통해 데이터 구조를 정의하며, 주로 다음과 같은 특징이 있습니다.

* **간결성**: 태그나 괄호 없이 간단한 텍스트 기반 형식.
* **유연성**: 다양한 데이터 타입과 구조를 지원.
* **사람이 읽기 쉬움**: 들여쓰기로 계층 구조를 표현.

### 3.2 주요 필드

이 작성법은 DevSecOps 환경에서 사용되는 주요 필드(apiVersion, kind, metadata, spec)와 Kubernetes 워커 노드를 추가하는 방법을 중심으로 작성되었습니다.

1) apiVersion

* 뜻: API 버전을 지정합니다. Kubernetes 리소스 종류마다 지원하는 API 버전이 다를 수 있습니다.
* 사용 목적: Kubernetes가 리소스를 처리할 때 올바른 API를 호출하도록 지시합니다.
* 예시: 여기서 apps/v1은 Deployment와 같은 애플리케이션 관련 리소스를 정의할 때 사용합니다.

apiVersion: apps/v1

2) kind

* 뜻: YAML 파일이 정의하는 리소스의 유형을 지정합니다.
* 사용 목적: Kubernetes에서 정의된 리소스 종류(Deployment, Pod 등)를 나타냅니다.
* 예시: 여기서 Deployment는 애플리케이션 배포를 정의함을 의미합니다.

kind: Deployment

3) metadata

* 뜻: 리소스에 대한 메타데이터를 정의합니다. 리소스 이름, 라벨, 주석 등을 포함합니다.
* 사용 목적: 리소스를 식별하거나 관리할 때 사용되는 정보를 제공합니다.
* 주요 필드:
  + name: 리소스의 이름
  + labels: 리소스에 태그를 추가하여 그룹화하거나 필터링에 사용
* 예시

metadata:

name: exmple-deployment

labels:

app: example

4) spec

* 뜻: 리소스의 동작 방식과 구성 세부 사항을 정의합니다.
* 사용 목적: Pod 탬플릿, 컨테이너 정의, 서비스 포트 등을 지정합니다.
* 주요 필드:
  + replicas: Pod 복제본 수
  + containers: 실행할 컨테이너 이미지와 설정
* 예시

spec:

replicas: 3

selector:

matchLabels:

app: example

template:

metadata:

labels:

app: example

spec:

containers:

- name: nginx

image: nginx:1.21.6

ports:

- containerPort: 80

### 3.3 워커 노드 추가

1) kind-config.yaml 파일

워커 노드 설정은 nodes 필드에 추가합니다.

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

nodes:

- role: control-plane

- role: worker

labels:

node-type: webserver

purpose: apache-php

- role: worker

labels:

node-type: database

purpose: mysql

* role: worker: 워커 노드로 역할을 지정
* labels: 각 워커 노드에 태그를 추가하여 특정 워크로드에 연결

2) Deployment에서 워커 노드 참조

특정 노드에서 실행되도록 nodeSelector를 사용합니다.

spec:

template:

spec:

nodeSelector:

node-type: webserver

3) 노드 상태 확인

Windows PowerShell 실행 후, 명령어를 통해 클러스트 상태 확인.

1. kubectl get nodes -–show-labels

### 3.4 기타 주요 필드

1) status

* 뜻: 리소스의 현재 상태를 나타냅니다.
* 사용 목적: 리소스가 배포된 후 Kubernetes가 관리하는 동적인 상태 정보를 기록합니다. 사용자가 수동으로 설정할 필요는 없습니다.
* 예시: 현재 3개의 Pod 중 2개가 정삭적으로 실행되고 있음을 나타냅니다.

status:

replicas: 3

availableReplicas: 2

2) selector

* 뜻: 리소스를 특정 레이블을 기준으로 연결합니다.
* 사용 목적: Deployment와 같은 리소스가 Pod를 선택하거나 Service가 대상 Pod를 식별할 때 사용됩니다.
* 예시: app: example 레이블이 포함된 리소스를 선택합니다.

selector:

matchLabels:

app: example

3) template

* 뜻: 리소스가 생성할 객체(Pod 등)의 템플릿을 정의합니다.
* 사용 목적: Deployment, DaemonSet 등의 리소스에서 사용되며, 생성할 Pod의 세부 설정을 포함합니다.
* 예시: app: example 레이블을 가진 Pod를 생성하며, nginx 컨테이너를 실행합니다.

template:

metadata:

labels:

app: example

spec:

containers:

- name: nginx

image: nginx:1.21.6

4) volumes

* 뜻: 컨테이너에서 사용할 볼륨을 정의합니다.
* 사용 목적: 데이터의 영속성을 보장하거나 컨테이너 간 데이터를 공유할 때 사용됩니다.
* 예시: Persistent Volume Claim(PVC)을 참조하여 데이터를 영구 저장합니다.

volumes:

- name: example-volume

persistentVolumeClaim:

claimName: example-pvc

5) resources

* 뜻: 컨테이너에서 사용할 CPU와 메모리 리소스를 제한하거나 요청합니다.
* 사용 목적: 클러스터의 리소스를 효율적으로 관리합니다.
* 예시:
  + limits: 컨테이너가 사용할 수 있는 최대 리소스 설정
  + requests: 컨테이너가 요청하는 최소 리소스 설정

resources:

limits:

memory: "512Mi"

cpu: "500m"

requests:

memory: "256Mi"

cpu: "250m"

6) env

* 뜻: 컨테이너 내에서 사용할 환경 변수를 정의합니다.
* 사용 목적: 동적인 환경 설정을 제공하거나 비밀 값을 주입합니다.
* 예시: 데이터베이스 호스트와 포트를 환경 변수로 정의합니다.

env:

- name: DB\_HOST

value: "example-database"

- name: DB\_PORT

value: "3306"

7) ports

* 뜻: 컨테이너가 노출하는 네트워크 포트를 정의합니다.
* 사용 목적: 외부 네트워크와의 통신 경로를 설정합니다.
* 예시: 컨테이너의 80번 포트를 외부로 노출합니다.

ports:

- containerPort: 80

8) nodeSelector

* 뜻: 특정 노드에서 리소스를 실행하도록 지정합니다.
* 사용 목적: 지정된 레이블을 기준으로 Pod가 실행될 노드를 선택합니다.
* 예시: node-type=database 레이블이 설정된 노드에서 Pod가 실행되도록 지정합니다.

nodeSelector:

node-type: database

9) affinity

* 뜻: Pod 배치 전략을 정의합니다.
* 사용 목적: 특정 노드에 Pod을 배치하거나, 특정 노드에서 실행되지 않도록 제약 조건을 설정합니다.
* 예시: node-type=webserver 노드에만 Pod가 배치되도록 지정합니다.

affinity:

nodeAffinity:

requiredDuringSchedulingIgnoredDuringExecution:

nodeSelectorTerms:

- matchExpressions:

- key: node-type

operator: In

values:

- webserver

# 4. Kubernates

### 4.1 kind

1) Kind 설치 및 환경설정

Window PowerShell 관리자 권한 실행

# KIND 다운로드

1. Invoke-WebRequest -Uri "https://kind.sigs.k8s.io/dl/v0.20.0/kind-windows-amd64" -OutFile "$env:USERPROFILE\kind.exe"

# bin 디렉토리 생성 및 이동

1. New-Item -Path "$env:USERPROFILE\bin" -ItemType Directory -Force
2. Move-Item -Path "$env:USERPROFILE\kind.exe" -Destination "$env:USERPROFILE\bin"

# 환경변수 설정

1. $env:Path += ";$env:USERPROFILE\bin"
2. [Environment]::SetEnvironmentVariable('Path', $env:Path, 'User')

# 설치 확인

1. kind –-version

2) kind-config.yaml 생성

kind-config.yaml 파일은 쿠버네티스에서 어떠한 클러스터를 생성할지 설정하는 파일이다.

DevSecOps폴더에 k8s폴더 생성 및 kind-config.yaml 파일 생성 후, 아래 코드 작성

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

nodes:

# 컨트롤 플레인

- role: control-plane

extraPortMappings:

- containerPort: 30080 #DevSecOps Web

hostPort: 30080

listenAddress: "0.0.0.0"

protocol: TCP

# 워커 노드 1: 테스트 웹서버

- role: worker

labels:

node-type: webserver

purpose: apache-php

3) 클러스터 생성 및 학인

Window PowerShell 관리자 권한 실행

※ Docker Desktop 실행 필요

# DevSecOps\k8s폴더로 이동

#(컴퓨터 설정에 따라 Users폴더 안 사용자 폴더이름 변경 필요)

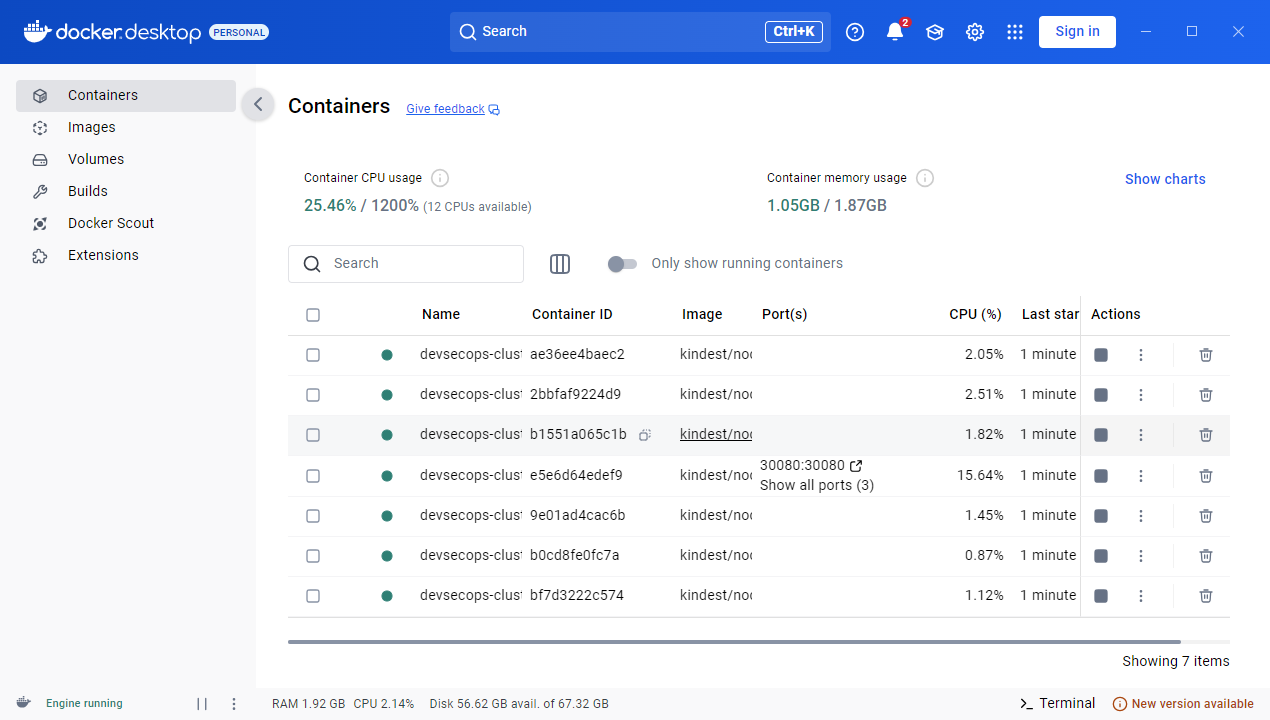
1. cd C:\Users\Administrator\Desktop\DevSecOps\k8s\

# kind 명령어로 새 클러스터 생성

1. kind create cluster --config kind-config.yaml --name devsecops-cluster

# 클러스터 생성 확인

1. kubectl get nodes --show-labels



Docker Desktop 화면

kind-config.yaml 파일에서 설정한 각 노드들이 컨테이너로 실행되고 있는 것을 볼 수 있다.

### 4.2 오류 발생 또는 충돌 시

1) 클러스터 삭제 및 Docker Desktop 재실행

Window PowerShell 관리자 권한 실행

# 모든 클러스터 삭제

1. kind delete clusters -–all

2) Docker Desktop 재실행 후, kindest/node 이미지 삭제

1. Docker Desktop 해당 이미지의 Delete 버튼(휴지통) 클릭



2. Docker Desktop 터미널에서 명령어 입력

# Docker Image 삭제

1. docker rmi <Image ID>

# 5. Jenkins

### 5.1 Jenkins 란?

Jenkins는 CI/CD를 위한 오픈소스로 도구로, 다양한 플러그인을 지원하여 코드 변경 사항이 발생할 때마다 **자동으로 빌드, 테스트, 배포 과정을 수행할 수 있는 도구**입니다.  
이를 통해 개발팀은 소프트웨어 개발 및 배포 과정을 효율적으로 관리할 수 있으며, 반복 작업을 자동화하여 개발 생산성을 크게 향상시킬 수 있습니다.

**CI/CD란 무엇인가?**

* **CI (지속적 통합, Continuous Integration)**: 개발자들이 각자 작업한 코드 변경 사항을 중앙 저장소에 정기적으로 통합하고, 이를 자동화된 빌드와 테스트를 통해 검증하는 프로세스이다. 이를 통해 코드 품질을 유지하고, 통합 시 발생할 수 있는 문제를 조기에 발견할 수 있습니다.
* **CD (지속적 배포, Continuous Deployment)**: CI 과정에서 통합된 코드가 자동으로 프로덕션 환경에 배포되기까지의 전 과정을 자동화하는 것을 의미합니다. 이로 인해 새로운 기능과 수정 사항이 사용자에게 빠르고 안정적으로 전달될 수 있습니다.

Jenkins는 이러한 CI/CD 프로세스를 손쉽게 구축하고 관리할 수 있는 도구로, 소프트웨어 개발과 배포의 효율성을 크게 향상합니다.

### 5.2 DevSecOps에서의 Jenkins의 역할

Jenkins는 DevSecOps 환경에서 **자동화의 핵심 도구**로 활용됩니다.

* 코드 변경 시 **자동으로 빌드하고 테스트하여 오류를 빠르게 발견**할 수 있습니다.
* **수천 개의 플러그인을 통해 다양한 기능**을 추가하여 복잡한 요구 사항을 충족시킬 수 있습니다.
* **배포 파이프라인을 코드로 정의하고 관리**할 수 있어, 재사용성과 유지보수성을 극대화합니다.
* **보안 검증, 취약점 스캔**과 같은 DevSecOps 요구사항도 Jenkins 파이프라인에 통합하여 자동화할 수 있습니다.

이러한 기능을 통해 Jenkins는 DevSecOps의 핵심 원칙인 **자동화, 통합, 보안**을 구현하는 데 중요한 역할을 합니다.

### 5.3 Jenkins worker node 생성

1) jenkins-deployment.yaml 생성

jenkins-deployment.yaml 파일은 jenkins worke node 설정 파일이다.

k8s-yaml 폴더에 jenkins-deployment.yaml 파일 생성 후, 아래 코드 작성

apiVersion: apps/v1

kind: Deployment

metadata:

name: jenkins

labels:

app: jenkins

spec:

replicas: 1

selector:

matchLabels:

app: jenkins

template:

metadata:

labels:

app: jenkins

spec:

nodeSelector:

purpose: jenkins

securityContext:

runAsUser: 0

fsGroup: 0

initContainers:

- name: init-jenkins-home

image: busybox

command: ['sh', '-c', 'mkdir -p /var/jenkins\_home && chown -R 1000:1000 /var/jenkins\_home']

volumeMounts:

- name: jenkins-home

mountPath: /var/jenkins\_home

containers:

- name: jenkins

image: jenkins/jenkins:lts

ports:

- containerPort: 8080

name: http

- containerPort: 50000

name: jnlp

volumeMounts:

- name: jenkins-home

mountPath: /var/jenkins\_home

env:

- name: JAVA\_OPTS

value: "-Xmx2048m"

resources:

limits:

memory: "2Gi"

cpu: "1000m"

requests:

memory: "1Gi"

cpu: "500m"

startupProbe:

httpGet:

path: /login

port: 8080

failureThreshold: 30

periodSeconds: 10

readinessProbe:

httpGet:

path: /login

port: 8080

initialDelaySeconds: 60

timeoutSeconds: 5

periodSeconds: 10

livenessProbe:

httpGet:

path: /login

port: 8080

initialDelaySeconds: 60

timeoutSeconds: 5

periodSeconds: 10

volumes:

- name: jenkins-home

persistentVolumeClaim:

claimName: jenkins-pvc

2) kind-config.yaml 수정

Jenkins 클러스터를 생성하기 위해 kind-config.yaml 파일 수정, 아래 코드 추가

kind: Cluster

apiVersion: kind.x-k8s.io/v1alpha4

nodes:

# 컨트롤 플레인

# Jenkins 웹 인터페이스 포트

# - CI/CD 파이프라인 관리 웹 UI

# - 빌드 및 배포 모니터링

# - 호스트 접근: http://localhost:8080

- containerPort: 30800

hostPort: 8080

listenAddress: "0.0.0.0"

protocol: TCP

# Jenkins JNLP 에이전트 포트

# - Jenkins 워커 노드 연결

# - 분산 빌드 에이전트 통신

# - 호스트 접근: localhost:50000

- containerPort: 30850

hostPort: 50000

listenAddress: "0.0.0.0"

protocol: TCP

# 워커 노드 2: Jenkins

- role: worker

labels:

node-type: auto

purpose: jenkins

3) Jenkins-pv.yaml 생성

PV(Persistent Volume)와 PVC(Claim)은 쿠버네티스에서 영구 스토리지를 관리하기 위한 핵심 리소스이다.

apiVersion: v1

kind: PersistentVolume

metadata:

name: jenkins-pv

labels:

type: local

spec:

storageClassName: manual

capacity:

storage: 10Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/data/jenkins-volume"

---

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: jenkins-pvc

spec:

storageClassName: manual

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 10Gi

4) Jenkins-service.yaml 생성

jenkins worker node의 연동을 위한 파일

apiVersion: v1

kind: Service

metadata:

name: jenkins

labels:

app: jenkins

spec:

type: NodePort

ports:

- port: 8080

targetPort: 8080

nodePort: 30800

name: http

- port: 50000

targetPort: 50000

nodePort: 30850

name: jnlp

selector:

app: Jenkins

### 5.4 Jenkins worker node 실행

1) kind 클러스터 재생성

Window PowerShell 관리자 권한 실행

# DevSecOps폴더로 이동

1. cd C:\Users\Administrator\Desktop\DevSecOps

# kind 클러스터 생성

1. kind create cluster --name devsecops-cluster --config k8s-yaml/kind-config.yaml

# 노드 상태 확인

1. kubectl get nodes --show-labels

디플로이먼트와 서비스 배포

# deployment, service yaml 파일 배포

1. kubectl apply -f k8s-yaml/jenkins-deployment.yaml
2. kubectl apply -f k8s-yaml/jenkins-services.yaml

# 배포 상태 확인

1. kubectl get pods -o wide
2. kubectl get services

2) Jenkins 관리자 암호

Jenkins worker node로부터 초기 관리자 암호 추출

# Jenkins Administrator Password에 필요한 패스워드값 출력

1. kubectl exec -it $(kubectl get pods -l app=jenkins -o jsonpath='{.items[0].metadata.name}') -- cat /var/jenkins\_home/secrets/initialAdminPassword

3) 웹 페이지 접속

browser 주소창: localhost:8080 or <http://Windows> Host IP:8080

# 6. ELK

### 6.1 ELK 란?

**ELK 스택**은 데이터 로그 및 데이터를 수집, 저장, 분석, 시각화하는 오픈소스 도구 세트입니다. Elasticsearch, Logstash, Kibana라는 세 가지 구성 요소로 이루어져 있으며, 이를 통해 로그 데이터를 실시간으로 모니터링하고, 문제를 빠르게 분석할 수 있습니다. 데이터를 시각적으로 표현하기 때문에 비즈니스 및 기술 운영의 효율성을 크게 향상시킬 수 있습니다.

* **Elasticsearch**: 데이터를 저장하고 빠르게 검색할 수 있는 **검색 및 분석 엔진**입니다. 대량의 데이터를 효율적으로 관리하고, 필요한 정보를 실시간으로 검색할 수 있습니다.
* **Logstash**: 데이터를 **수집하고 변환**하는 파이프라인 도구입니다. 다양한 데이터 소스에서 로그 데이터를 가져와 정리한 후 Elasticsearch에 전달합니다.
* **Kibana**: 데이터를 **시각화**하는 인터페이스입니다. Elasticsearch에 저장된 데이터를 기반으로 차트, 그래프, 대시보드를 생성하여 쉽게 분석할 수 있도록 도와줍니다.

### 6.2 ELK worker node 생성

1) elasticsearch.yaml 생성

elasticsearch.yaml 파일은 **Elasticsearch**를 **Kubernetes** 환경에 배포하기 위한 **Deployment와 Service**를 정의한 파일이다. 파일의 각 섹션에 대한 자세한 설명은 다음과 같다.

apiVersion: apps/v1

kind: **Deployment**

metadata:

name: elasticsearch

labels:

app: elasticsearch

spec:

selector:

matchLabels:

app: elasticsearch

template:

metadata:

labels:

app: elasticsearch

spec:

nodeSelector:

purpose: elk-stack

containers:

- name: elasticsearch

image: docker.elastic.co/elasticsearch/elasticsearch:8.10.2

env:

- name: discovery.type

value: single-node

- name: xpack.security.enabled

value: "false"

- name: xpack.security.http.ssl.enabled

value: "false"

- name: network.host

value: "0.0.0.0"

ports:

- containerPort: 9200

name: http

protocol: TCP

---

apiVersion: v1

kind: Service

metadata:

name: elasticsearch

labels:

app: elasticsearch

spec:

type: NodePort

selector:

app: elasticsearch

ports:

- port: 9200

targetPort: 9200

nodePort: 30920

name: http

protocol: TCP

2) kibana.yaml 생성

kibana.yaml 파일은 Kibana 애플리케이션을 Kubernetes 클러스터에 배포하고 외부에서 접근할 수 있도록 구성된 YAML 파일입니다. 주요 구성 요소는 Deployment와 Service입니다. 각 섹션을 아래에서 자세히 설명합니다.

apiVersion: apps/v1

kind: Deployment

metadata:

name: kibana

labels:

app: kibana

spec:

selector:

matchLabels:

app: kibana

template:

metadata:

labels:

app: kibana

spec:

nodeSelector:

purpose: elk-stack

containers:

- name: kibana

image: docker.elastic.co/kibana/kibana:8.10.2

env:

- name: ELASTICSEARCH\_HOSTS

value: http://elasticsearch:9200

- name: SERVER\_HOST

value: "0.0.0.0"

ports:

- containerPort: 5601

name: http

protocol: TCP

---

apiVersion: v1

kind: Service

metadata:

name: kibana

labels:

app: kibana

spec:

type: NodePort

selector:

app: kibana

ports:

- port: 5601

targetPort: 5601

nodePort: 30601

name: http

protocol: TCP

3) logstash.yaml 생성

apiVersion: v1

kind: ConfigMap

metadata:

name: logstash-config

data:

logstash.conf: |

input {

beats {

port => 5044

}

}

output {

elasticsearch {

hosts => ["elasticsearch:9200"]

index => "%{[@metadata][beat]}-%{[@metadata][version]}-%{+YYYY.MM.dd}"

}

}

---

apiVersion: apps/v1

kind: Deployment

metadata:

name: logstash

labels:

app: logstash

spec:

selector:

matchLabels:

app: logstash

template:

metadata:

labels:

app: logstash

spec:

nodeSelector:

purpose: elk-stack

containers:

- name: logstash

image: docker.elastic.co/logstash/logstash:8.10.2

volumeMounts:

- name: config-volume

mountPath: /usr/share/logstash/pipeline/logstash.conf

subPath: logstash.conf

ports:

- containerPort: 5044

volumes:

- name: config-volume

configMap:

name: logstash-config

---

apiVersion: v1

kind: Service

metadata:

name: logstash

labels:

app: logstash

spec:

selector:

app: logstash

ports:

- port: 5044

targetPort: 5044

# 7. Web 구축

### 7.1. WebPage 소스코드

방법 1) 직접 생성

1. DevSecOps폴더에 Web폴더 생성 후, 웹 소스코드 복사

Practice Web Source code: [DevSecOps Team Web Source code](https://github.com/GH6679/web_wargamer)

2. Github의 특수문자를 허용하기 위한 설정

Window PowerShell 관리자 권한 실행

# DevSecOps\Web폴더로 이동

1. cd C:\Users\Administrator\Desktop\DevSecOps\Web

# Github 특수문자 허용

2. git config core.protectNTFS false

3. DevSecOps폴더에 k8s-yaml 폴더 생성

방법 2) Window PowerShell 사용

Window PowerShell 관리자 권한 실행

# DevSecOps폴더 이동

1. cd C:\Users\Administrator\Desktop\DevSecOps

# 소스코드 클론

2. git clone <https://github.com/GH6679/web_wargamer.git>

3. cd web\_wargamer

# 깃허브 안에 있는 특수문자들 허용으로 설정

4. git config core.protectNTFS false

5. cd ..

# k8s-yaml 디렉토리 생성

6. mkdir k8s-yaml

### 7.2. Web worker node 생성

1) web-deployment.yaml 생성

web-deployment.yaml 파일은 웹 worker node 설정 파일이다.

k8s-yaml 폴더에 web-deployment.yaml 파일 생성 후, 아래 코드 작성

apiVersion: apps/v1

kind: Deployment

metadata:

name: wargame-web

spec:

selector:

matchLabels:

app: wargame-web

template:

metadata:

labels:

app: wargame-web

node-type: webserver

spec:

nodeSelector:

node-type: webserver

containers:

- name: wargame-web

image: wargame-web:latest

imagePullPolicy: Never

ports:

- containerPort: 80

env:

- name: DB\_HOST

value: "wargame-db-service"

- name: DB\_PORT

value: "3307"

- name: DB\_DATABASE

value: "LED\_WG"

- name: DB\_USERNAME

value: "root"

- name: DB\_PASSWORD

value: "1234"

volumeMounts:

- name: web-storage

mountPath: /var/www/html

resources:

limits:

memory: "512Mi"

cpu: "500m"

requests:

memory: "256Mi"

cpu: "250m"

volumes:

- name: web-storage

persistentVolumeClaim:

claimName: web-pvc

2) db-deployment.yaml 생성

db-deployment.yaml 파일은 웹 데이터베이스 worker node 설정 파일이다.

k8s-yaml 폴더에 db-deployment.yaml 파일 생성 후, 아래 코드 작성

apiVersion: apps/v1

kind: Deployment

metadata:

name: wargame-db

spec:

selector:

matchLabels:

app: wargame-db

template:

metadata:

labels:

app: wargame-db

node-type: web-db

spec:

nodeSelector:

node-type: web-db

containers:

- name: mariadb

image: mariadb:latest

resources:

limits:

memory: "512Mi"

cpu: "500m"

ports:

- containerPort: 3306

env:

- name: MYSQL\_ROOT\_PASSWORD

value: "1234"

- name: MYSQL\_DATABASE

value: "LED\_WG"

- name: MYSQL\_ALLOW\_EMPTY\_PASSWORD

value: "no"

- name: MYSQL\_ROOT\_HOST

value: "%"

- name: MARIADB\_MYSQL\_LOCALHOST\_USER

value: "1"

- name: MARIADB\_MYSQL\_LOCALHOST\_GRANTS

value: "ALL"

volumeMounts:

- name: mariadb-storage

mountPath: /var/lib/mysql

- name: mariadb-backup

mountPath: /backup

volumes:

- name: mariadb-storage

persistentVolumeClaim:

claimName: mariadb-pvc

- name: mariadb-backup

persistentVolumeClaim:

claimName: mariadb-backup-pvc

3) db-init-configmap.yaml 생성

db-init-configmap.yaml 파일은 웹 데이터베이스 worker node의 초기화 설정 파일이다.

k8s-yaml 폴더에 db-init-configmap.yaml 파일 생성 후, 아래 코드 작성

apiVersion: v1

kind: ConfigMap

metadata:

name: mariadb-init

data:

init.sql: |

USE LED\_WG;

CREATE TABLE IF NOT EXISTS users (

u\_id INT NOT NULL AUTO\_INCREMENT,

nickname VARCHAR(50) NOT NULL,

username VARCHAR(50) NOT NULL,

password VARCHAR(50) NOT NULL,

email VARCHAR(50) NOT NULL,

user\_role VARCHAR(50) NOT NULL,

PRIMARY KEY (u\_id)

);

CREATE TABLE IF NOT EXISTS challenges\_data (

c\_id INT NOT NULL AUTO\_INCREMENT,

c\_title VARCHAR(50) NOT NULL,

c\_ssh VARCHAR(50),

c\_web TINYINT(1) NOT NULL,

c\_link VARCHAR(50),

c\_hint TEXT,

c\_point int,

c\_difficulty VARCHAR(50),

c\_key VARCHAR(50),

c\_text TEXT,

c\_solves int,

PRIMARY KEY (c\_id)

);

CREATE TABLE IF NOT EXISTS user\_data (

d\_id INT NOT NULL AUTO\_INCREMENT,

d\_uid INT,

d\_cid INT,

d\_time datetime,

PRIMARY KEY (d\_id),

FOREIGN KEY (d\_uid) REFERENCES users(u\_id) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (d\_cid) REFERENCES challenges\_data(c\_id) ON DELETE CASCADE ON UPDATE CASCADE);

4) service.yaml 생성

service.yaml 파일은 웹과 데이터베이스의 연동을 위한 설정 파일이다.

k8s-yaml 폴더에 service.yaml 파일 생성 후, 아래 코드 작성

apiVersion: v1

kind: Service

metadata:

name: wargame-web-service

spec:

type: NodePort

ports:

- port: 80

targetPort: 80

nodePort: 30080

selector:

app: wargame-web

---

apiVersion: v1

kind: Service

metadata:

name: wargame-db-service

spec:

type: ClusterIP

ports:

- port: 3306

targetPort: 3306

protocol: TCP

selector:

app: wargame-db

### 7.3 Web worker node 실행

1) 클러스터 삭제 및 kind 클러스터 생성

Window PowerShell 관리자 권한 실행

# DevSecOps폴더로 이동

1. cd C:\Users\Administrator\Desktop\DevSecOps

# 클러스터 삭제 명령어

2. kind delete cluster --name devsecops-cluster

3. kind delete clusters -–all

# kind 클러스터 생성

4. kind create cluster --name devsecops-cluster --config k8s-yaml/kind-config.yaml

# 노드 상태 확인

5. kubectl get nodes --show-labels

Web 이미지 빌드 및 kind 클러스터로 로드

# 웹 애플리케이션 이미지 빌드

1. docker build -t wargame-web:latest .

# 빌드된 이미지를 kind 클러스터로 로드

2. kind load docker-image wargame-web:latest --name devsecops-cluster

디플로이먼트와 서비스 배포

# deployment, service yaml 파일 배포

1. kubectl apply -f k8s-yaml/db-deployment.yaml

2. kubectl apply -f k8s-yaml/web-deployment.yaml

3. kubectl apply -f k8s-yaml/services.yaml

# 배포 상태 확인

4. kubectl get pods -o wide

5. kubectl get services

2) 웹 페이지 접속

browser 주소창: localhost:30080 or <http://Windows> Host IP:30080