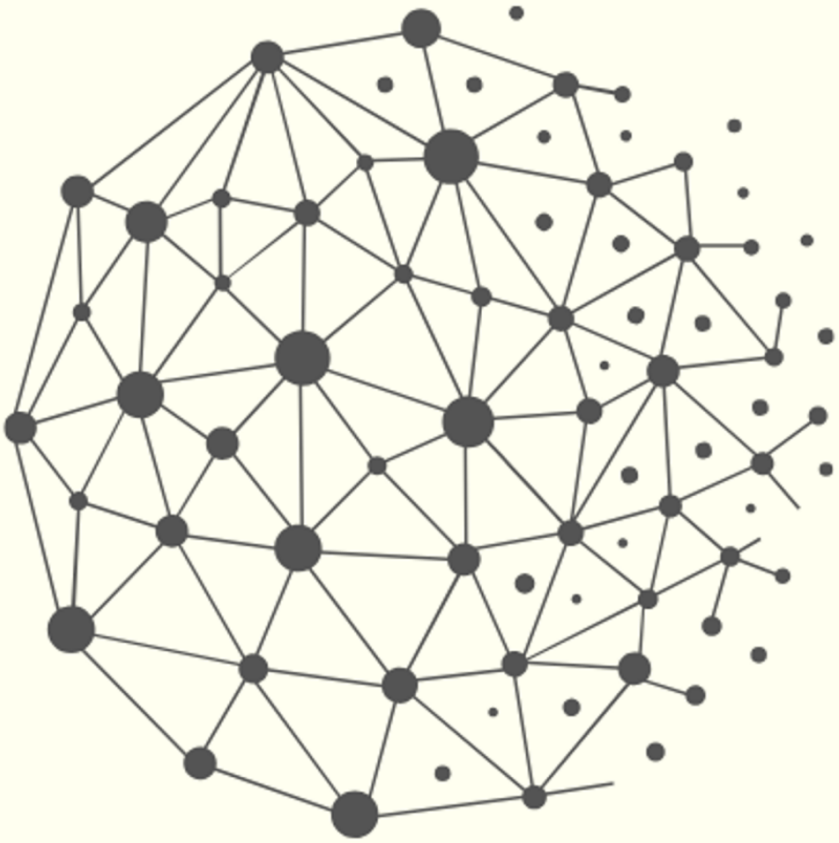


2025 DevSecOps 환경 구축 가이드

Nova

TEAM-DEVSECOPS

서론

2025 DevSecOps 환경 구축 가이드는 DevSecOps 방식을 통해 개발, 운영, 보안을 효과적으로 통합하여 CI/CD 프로세스를 최적화하고 시스템의 안전성을 강화하는 방법을 제시합니다. 특히, Jenkins, SonarQube, Docker, Kubernetes 등을 활용하여 소스 코드 품질을 개선하고 보안 자동화를 구현하는 구체적인 접근법을 제공합니다.

이 문서는 다음을 목표로 합니다.

1. **개발성 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. **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

기획팀

정재호 장진영 박진우 안정훈 지윤정

개발팀

배준성 윤광혁 최맨재 장태경

운영팀

김범준 김지홍 노윤서 손혜영 이주원

Nova

TEAM-DEVSECOPS

Project Git Repository

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

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

목차

1. DevSecOps

1.1. DevSecOps 란?

DevSecOps 는 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 DevSecOps 는 초기 설계 단계부터 배포, 운영에 이르기까지 보안을 모든 과정에 포함시키며, 이를 통해 **보안은 모두의 책임**이라는 문화를 강조합니다.

DevSecOps 의 주요 원칙은 다음과 같습니다.

- **자동화된 보안 통합**: CI/CD 파이프라인에 보안 도구를 통합하여 코드 분석, 취약점 점검, 종속성 관리를 자동화합니다.
- **보안 중심의 협업**: 개발, 운영, 보안 팀이 협력하여 보안 문제를 조기에 발견하고 해결합니다.
- **지속적인 개선**: 위협 모델링과 실시간 모니터링으로 보안을 지속적으로 강화합니다.

우리는 DevSecOps 구현을 위해 Jenkins 와 같은 CI/CD 도구를 기반으로 SonarQube 로 코드 품질 및 보안 점검을 수행하고 시스템 로그를 분석하여 이상 행동을 탐지합니다. 또한, Docker 와 Kubernetes 로 컨테이너 보안을 강화하여 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 DevSecOps 의 이점

DevSecOps 를 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, DevOps 의 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.

DevSecOps 에서는 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 DevSecOps 를 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.

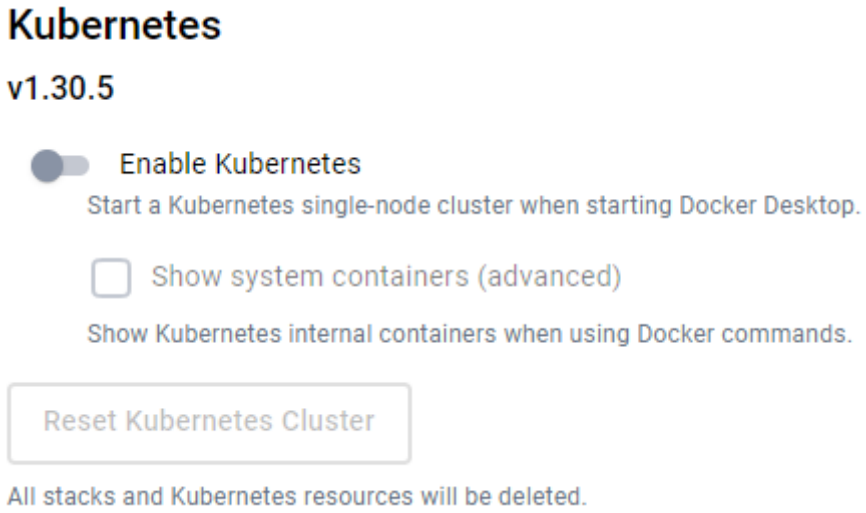
2. 사전 준비 사항

2.1 사전 요구 사항 (필요 설치 프로그램)

프로그램	설명
WSL	Windows 에서 Linux 환경을 구축할 수 있게 해주는 핵심 기능을 한다.
Docker Desktop(kind) Kuberctl	Kind(Kubernetes IN Docker)는 Docker 컨테이너를 사용하여 Kubernetes 클러스터를 쉽게 배포하고 관리할 수 있게 해주는 도구이다.
Git	Github 및 Jenkins 를 통한 자동화 핵심 기능을 한다.
Chocolatey	Chocolatey 는 Windows 용 패키지 관리자 도구이다.
localtunnel	로컬 서버를 인터넷에서 접근 가능한 URL 로 노출시켜주는 도구이다.

본 프로젝트의 작업은 Windows 운영체제의 바탕화면에서 진행됩니다.

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



Docker Desktop 설정 화면

Kubernetes > Enable Kubernetes 의 설정이 해제되어 있는 것을 볼 수 있다.

2.2 사전 요구 사항 설치

Window PowerShell 관리자 권한으로 실행

1) WSL(Windows Subsystem for Linux) 설치

프로그램	설명
WSL	Windows 에서 Linux 환경을 구축할 수 있게 해주는 핵심 기능을 한다.

설치 코드

```
# WSL 설치
$  dism.exe  /online  /enable-feature  /featurename:Microsoft-Windows-
   Subsystem-Linux /all /norestart
$  dism.exe /online /enable-feature /featurename:VirtualMachinePlatform
   /all /norestart

# WSK2 기본 버전으로 설정
$  wsl -set-default-version 2
```

2) Chocolatey 설치

프로그램	설명
Chocolatey	Chocolatey 는 Windows 용 패키지 관리자 도구이다.

설치 코드

```
# Chocolatey 패키지 관리자 설치
$  Set-ExecutionPolicy Bypass -Scope Process -Force
$  [System.Net.ServicePointManager]::SecurityProtocol =
   [System.Net.ServicePointManager]::SecurityProtocol -bor 3072
$  3. iex ((New-Object
   System.Net.WebClient).DownloadString('https://community.chocolatey.or
   g/install.ps1'))
```

3) Git 설치

프로그램	설명
Git	Github 및 Jenkins 를 통한 자동화 핵심 기능을 한다.

설치 코드

```
# Git 기본 패키지 설치
$  choco install git -y
```

4) Docker Desktop, kubectl 및 kind 설치

프로그램	설명
1. Docker **2. kuctl** **3. kind**	Kind(Kubernetes IN Docker)는 Docker 컨테이너를 사용하여 Kubernetes 클러스터를 쉽게 배포하고 관리할 수 있게 해주는 도구이다.

설치 코드

```
# Docker Desktop 설치
$  choco install docker-desktop -y

# kubectl 과 kind 설치
$  choco install kubernetes-cli -y
$  choco install kind -y
```

5) Localtunnel 설치

프로그램	설명
Localtunnel	로컬 서버를 인터넷에서 접근 가능한 URL 로 노출시켜주는 도구이다.

설치 코드

```
# node.js 다운로드
$  http://nodejs.org/ko/

# localtunnel 다운로드
$  npm install -g localtunnel

# Window cmd 관리자 권한 실행
$  lt -subdomain <도메인 이름> --port <사용될 포트>

# 터널 비밀번호
$  http://loca.lt/mytunnelpassword
```

3. yaml 작성법

3.1 yaml 이란?

YAML 은 "YAML Ain't Markup Language"의 약자로, 데이터 직렬화에 사용되는 가볍고 사람이 읽기 쉬운 형식입니다. JSON 이나 XML 과 비교해 간결하고 이해하기 쉬운 구문을 제공하며, 설정 파일이나 데이터 저장에 자주 활용됩니다. YAML 은 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

필드 이름	apiVersion
뜻	API 버전을 지정합니다. Kbuernetes 리소스 종류마다 지원하는 API 버전이 다를 수 있습니다.
사용 목적	Kubernetes 가 리소스를 처리할 때 올바른 API 를 호출하도록 지시합니다.
예시	여기서 apps/v1 은 Deployment 와 같은 애플리케이션 관련 리소스를 정의할 때 사용합니다.
`apiVersion: apps/v1`	

2) kind

필드 이름	kind
뜻	yaml 파일이 정의하는 리소스의 유형을 지정합니다.
사용 목적	Kubernetes 에서 정의된 리소스 종류를 나타냅니다.
예시	여기서 Deployment 는 애플리케이션 배포를 의미합니다.
`kind: Deployment`	

3) metadata

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

```
metadata:
      name: exmple-deployment
      labels: app: example
```


4) spec

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

```
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

파일	kind-config.yaml
사용 목적	kubernetes 노드에 어떠한 클러스터를 생성할지 설정하는 파일입니다.
예시	• role: worker: 워커 노드로 역할을 지정 • labels: 각 워커 노드에 태그를 추가하여 특정 워커노드에 연결

```
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
```

2) Deployment 에서 워커 노드 참조

파일	jenkins-deployment.yaml, web-deployment.yaml
사용 목적	특정 노드에서 실행되도록 nodeSelector 를 사용합니다.
예시	• role: worker: 워커 노드로 역할을 지정 • labels: 각 워커 노드에 태그를 추가하여 특정 워커노드에 연결

```
spec:
  template:
    spec:
      nodeSelector:
```

```
        node-type: webserver
```

3) 노드 상태 확인

Wiondows PowerShell 실행 후, 명령어를 통해 클러스트 상태 확인.

```
$  kubectl get nodes –show-labels
```

3.4 기타 필드

1) status

필드 이름	status
뜻	리소스의 현재 상태를 나타냅니다.
사용 목적	리소스가 배포된 후 Kubernetes 가 관리하는 동적인 상태 정보를 기록합니다. 사용자가 수동으로 설정할 필요는 없습니다.
예시	현재 3 개의 Pod 중 2 개가 정상적으로 실행되고 있음을 나타냅니다.

```
status:
  replicas: 3
  availableReplicas: 2
```


2) selector

필드 이름	selector
뜻	리소스를 특정 레이블을 기준으로 연결합니다.
사용 목적	Deployment 와 같은 리소스가 Pod 를 선택하거나 Service 가 대상 Pod 를 식별 할 때 사용됩니다.
예시	app: example 레이블이 포함된 리소스를 선택합니다.

```
selector:
  matchLabels:
    app: example
```

3) template

필드 이름	template

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

```
template:
  metadata:
    labels:
      app: example
  spec:
    containers:
    - name: nginx
      image: nginx:1.21.6
```

4) volumes

필드 이름	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

필드 이름	resources
뜻	컨테이너에서 사용할 CPU 와 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

필드 이름	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

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

```
ports:
- containerPort: 80
```

8) nodeSelector

필드 이름	nodeSelector
뜻	특정 노드에서 리소스를 실해ㅇ하도록 지정합니다.
사용 목적	지정된 레이블을 기준으로 Pod 가 실행될 노드를 선택합니다.
예시	node-type=database 레이블이 설정된 노드에서 Pod 가 실행되도록 지정합니다.

```
nodeSelector:
  node-type: database
```

8) affinity

필드 이름	affinity
뜻	Pod 배치 전략을 정의합니다.
사용 목적	특정 노드에 Pod 을 배치하거나, 특정 노드에서 실행되지 않도록 제약 조건을 설정합니다.
예시	node-type=webserver 노드에만 Pod 가 배치되도록 지정합니다.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: node-type
          operator: In
          values:
          - webserver
```

4. Kubernetes

4.1 Kubernetes 란?

Kubernetes(쿠버네티스)는 **컨테이너화된 애플리케이션을 쉽고 빠르게 배포 및 확장하며 관리를 자동화**해주는 오픈소스 오케스트레이션 플랫폼입니다. 애플리케이션의 안정적이고 효율적인 운영을 지원하며, 클라우드와 온프레미스 환경 모두에서 사용할 수 있습니다.

Kubernetes 는 다양한 구성 요소를 통해 컨테이너화된 애플리케이션을 관리합니다. 주요 개념은 다음과 같습니다.

Kubernetes 주요 개념		
No	개념	설명
1	컨테이너	• Docker 와 같은 컨테이너 기술로 애플리케이션을 격리된 환경에서 실행합니다. • Kubernetes 는 이러한 컨테이너의 배포와 관리를 담당합니다.
2	노드	• Kubernetes 클러스터를 구성하는 서버 단위입니다. • **마스터 노드**는 클러스터를 관리하고, **워커 노드**는 컨테이너를 실행합니다.
3	클러스터	• Kubernetes 가 관리하는 시스템 단위로 여러 노드로 구성됩니다. • 작업을 노드 간에 분배하여 효율적인 리소스 활용을 지원합니다.
4	파드	• Kubernetes 에서 가장 작은 배포 단위로, 하나 이상의 컨테이너가 포함됩니다. • 동일한 네트워크 및 스토리지를 공유하며 함께 작동합니다.
5	디플로이먼트	• 애플리케이션 배포 및 관리를 담당하며, 원하는 상태(예: 컨테이너의 수)를 정의합니다. • Kubernetes 는 이 상태를 유지하도록 자동으로 조정합니다.
6	서비스	파드 간의 통신을 가능하게 하며, 외부 네트워크와의 연결을 제공합니다.

7	이벤트 기반 관리	애플리케이션 상태를 지속적으로 모니터링하며, 필요 시 재배치, 재시작, 확장을 수행합니다.

따라서, 우리는 Kubernetes 의 주요개념을 기반으로 DevSecOps 환경을 효율적으로 관리할 계획입니다. 컨테이너를 통해 애플리케이션을 격리된 환경에서 실행하고, 노드와 클러스터를 활용해 리소스를 최적화하며 안정적인 시스템 운영을 지원합니다.

파드와 디플로이먼트를 이용해 애플리케이션을 유연하게 배포하고, 서비스를 통해 내부 및 외부 네트워크 통신을 원활히 처리합니다. 또한, Kubernetes 의 이벤트 기반 관리 기능을 활용해 애플리케이션 상태를 지속적으로 모니터링하고, 필요 시 자동으로 확장하거나 복구하는 방안을 적용할 것입니다.

4.2 DevSecOps 에서 Kubernetes 의 역할

우리가 Kubernetes 를 사용하는 이유는 DevSecOps 환경에서 애플리케이션을 효율적이고 안정적으로 관리하기 위함입니다. Kubernetes 는 리소스를 최적화하여 비용을 절감하고, 빠른 배포와 자동 복구를 통해 운영 안정성을 제공합니다.

또한, 클라우드와 온프레미스 환경을 모두 지원하며, 애플리케이션 확장과 멀티 클라우드 통합이 가능해 유연성과 확장성이 뛰어납니다. CI/CD 도구와의 손쉬운 통합으로 보안을 강화할 수 있어 우리의 프로젝트에 필수적인 플랫폼입니다.

Kubernetes 를 사용함으로써 DevSecOps 환경에서 생산성과 안정성을 동시에 향상시킬 수 있습니다. 자동화된 배포와 리소스 최적화를 통해 개발 및 운영 시간을 단축하고 비용을 절감할 수 있습니다. 장애 발생 시 자동 복구와 중단 없는 업데이트를 제공하여 시스템 가용성과 사용자 경험을 크게 개선합니다.

또한, 멀티 클라우드 및 하이브리드 환경을 지원해 유연한 인프라 구성이 가능하며, 애플리케이션의 확장성을 확보할 수 있습니다. Jenkins, SonarQube 등 다양한 DevSecOps 도구와의 통합으로 파이프라인 전반의 보안과 효율성을 강화할 수 있어,

전체적인 운영 관리가 간소화되고 보안 위험이 최소화됩니다. 이를 통해 안정적이고 신뢰할 수 있는 애플리케이션 서비스를 제공할 수 있습니다.

4.3 kind

우리는 kind(Kubernetes IN Docker)를 사용하여 클러스터를 구성하고 노드를 설치 및 관리합니다. kind 는 Docker 컨테이너 내에서 Kubernetes 클러스터를 실행할 수 있어, 간편하고 빠르게 개발 및 테스트 환경을 구축할 수 있다는 장점이 있습니다.

1) kind 설치 및 환경 설정

Window PowerShell 관리자 권한 실행

kind 설치 및 환경 설정 코드

```
# kind 다운로드
$  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
$  New-Item -Path "$env:USERPROFILE\bin" -ItemType Directory -Force
$  Move-Item     -Path     "$env:USERPROFILE\kind.exe"     -Destination
   "$env:USERPROFILE\bin"

# 환경 변수 설정
$  $env:Path += ";$env:USERPROFILE\bin"
$  [Environment]::SetEnvironmentVariable('Path', $env:Path, 'User')

# 설치 확인
$  kind –version
```

2) kind-config.yaml 생성

파 일 이 름	kind-config.yaml
파 일 설 명	쿠버네티스에서 어떠한 클러스터를 생성할지 설정하는 파일입니다.
파 일 코 드	DevSecOps 폴더에 k8s-yaml 폴더 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 확인

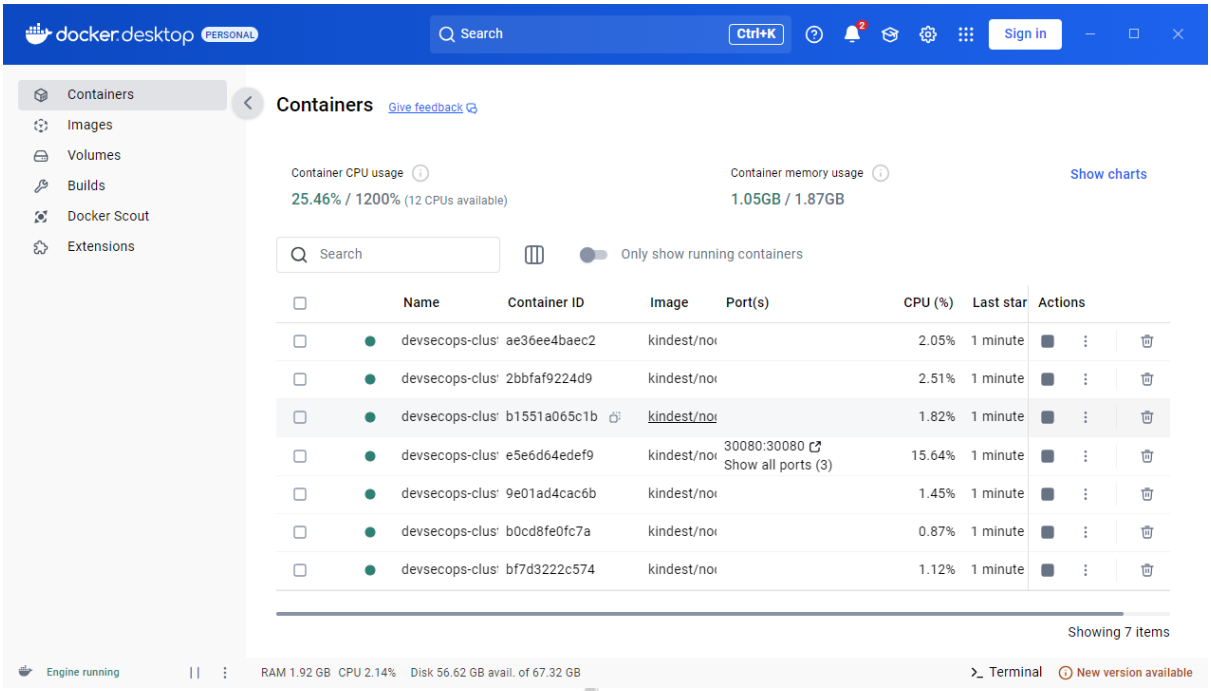
Window Powershell 관리자 권한 실행 (※ Docker Desktop 실행 필요)

클러스터 생성 및 확인 코드

```
# DevSecOps\k8s 폴더로 이동
#(컴퓨터 설정에 따라 Users 폴더 안 사용자 폴더이름 변경 필요)
$  cd C:\Users\Administrator\Desktop\DevSecOps\k8s\

# kind 명령어로 새 클러스터 생성
$  kind create cluster --config kind-config.yaml --name devsecops-cluster

# 클러스터 생성 확인
$  kind get clusters
```



Docker Desktop Containers 화면

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

4.4 오류 발생 또는 충돌 시

1) 클러스터 삭제 및 Docker Desktop 재실행

Window PowerShell 관리자 권한 실행 후 아래 명령어 입력

```
# 모든 클러스터 삭제
$  kind delete clusters --all
```

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

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

2. Docker Dekstop 터미널에서 아래 명령어 입력

```
# Docker Image 삭제
$  docker rmi <Image ID>
```

5. Jenkins

5.1 Jenkins 란?

Jenkins 는 CI/CD 를 위한 오픈소스로 도구로, 다양한 플러그인을 지원하여 코드 변경 사항이 발생할 때마다 자동으로 빌드, 테스트, 배포 과정을 수행할 수 있는 도구입니다. 이를 통해 개발팀은 소프트웨어 개발 및 배포 과정을 효율적으로 관리할 수 있으며, 반복 작업을 자동화하여 개발 생산성을 크게 향상시킬 수 있습니다.

CI/CD 란 무엇인가?

- **CI (지속적 통합, Continuous Intergration):** 개발자들이 각자 작업한 코드 변경 사항을 중앙 저장소에 정기적으로 통합하고, 이를 자동화된 빌드와 테스트를 통해 검증하는 프로세스입니다. 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.

Jenkins 는 이러한 CI/CD 프로세스를 구축하고 관리할 수 있는 도구로, 소프트웨어 개발과 배포의 효율성을 크게 향상시킵니다.

5.2 DevSecOps 에서의 Jenkins 의 역할

Jenkins 는 DevSecOps 환경에서 **자동화의 핵심** 도구로 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 Jenkins 는 DevSecOps 의 핵심 원칙인 자동화, 통합, 보안을 구현하는 데 중요한 역할을 수행합니다.

5.3 Jenkins worker node 생성

1) kind-config.yaml 수정

파 일 이 름	kind-config.yaml
파 일 설 명	쿠버네티스에서 어떠한 클러스터를 생성할지 설정하는 파일입니다.
파 일 코 드	kind-config.yaml 기존 코드 삭제 후, 아래 코드 작성

```
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
```

2) jenkins-deployment.yaml 생성

파 일 이 름	jenkins-deployment.yaml
파 일 설 명	jenkins worker node 의 설정 파일입니다.
파 일 코 드	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
```

3) jenkins-pv.yaml 생성

파 일 이 름	jenkins-pv.yaml
파 일 설 명	PV(Persistent Volume)와 PVC(Claim)은 쿠버네티스에서 영구 스토리지를 관리하기 위한 핵심 리소스입니다.
파 일 코 드	k8s-yaml 폴더에 jenkins-pv.yaml 파일 생성 후, 아래 코드 작성

```
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

파 일 이 름	jenkins-service.yaml
파 일 설 명	jenkins worker node 의 연동을 위한 파일입니다.
파 일 코 드	k8s-yaml 폴더에 jenkins-service.yaml 파일 생성 후, 아래 코드 작성

```
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

Window PowerShell 관리자 권한 실행 후, 명령어 입력

1) kind 클러스터 재생성

```
# DevSecOps 폴더로 이동
$  cd C:\Users\Administrator\Desktop\DevSecOps

# kind 클러스터 생성
$  kind create cluster --name devsecops-cluster --config k8s-yaml/kind-
   config.yaml

# 노드 상태 확인
```

```
$  kubectl get nodes --show-labels

# deployment, service.yaml 파일 배포
$  kubectl apply -f k8s-yaml/jenkins-deployment.yaml
$  2. kubectl apply -f k8s-yaml/jenkins-services.yaml

# 배포 상태 확인
$  kubectl get pods -o wide
$  kubectl get services
```

2) Jenkins 관리자 암호

jenkins worker node 로부터 초기 관리자 암호 추출

```
# Jenkins Administrator Password 에 필요한 패스워드값 출력
$  kubectl exec -it $(kubectl get pods -l app=jenkins -o
   jsonpath='{.items[0].metadata.name}') -- cat
   /var/jenkins_home/secrets/initialAdminPassword
```

3) 웹페이지 접속

```
Broser 주소창: localhost:8080 or http://Window Host IP:8080
```


4) localtunnel 을 이용한 도메인 연결

Window cmd 관리자 권한 실행

```
$  lt –subdomain jenkins1234 –port 8080
```

6. SonarQube

6.1 SonarQube 란?

SonarQube 는 지속적인 코드 품질 관리를 위한 오픈소스 정적 코드 분석 플랫폼입니다. 다양한 프로그래밍 언어를 지원하며, 코드의 품질, 버그, 취약점 등을 자동으로 분석하여 개발팀이 더 나은 소프트웨어를 만들 수 있도록 도와주는 도구입니다.

1) SonarQube 의 주요 기능

- **코드 품질 분석**: 소스 코드의 버그, 취약점을 감지하고, 중복 코드, 코딩 표준, 단위 테스트, 코드 커버리지, 복잡도 등을 분석하여 전반적인 코드 품질을 평가합니다.
- **지속적 통합**: CI/CD 파이프라인과 통합되어 자동화된 코드 분석을 제공하며, Jenkins, GitHub Actions 등 다양한 CI 도구와 연동이 가능합니다.
- **다중 환경 지원**: Java, Python, JavaScript, C# 등 30 개 이상의 프로그래밍 언어를 지원하며, Linux, Windows, Mac 등 다양한 환경에서 구동이 가능합니다.

SonarQube 는 이러한 기능들을 통해 개발팀이 코드 품질을 지속적으로 모니터링하고 개선할 수 있도록 도와주며, 소프트웨어의 전반적인 품질 향상에 크게 기여합니다.

6.2 SonarQube worker node 생성

DevSecOps 폴더에 sonarqube 폴더 생성 후, yaml 설정 파일 생성

1) postgres-deployment.yaml

파 일 이 름	postgres-deployment.yaml
파 일 설 명	sonar postgres worker node 의 설정 파일입니다.
파 일 코 드	postgres-deployment.yaml 파일 생성 후, 아래 코드 작성

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sonar-postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sonar-postgres
  template:
    metadata:
      labels:
        app: sonar-postgres
    spec:
      nodeSelector:
        node-type: jenkins
        purpose: jenkins
      containers:
      - name: postgres
        image: postgres:12
        env:
        - name: POSTGRES_DB
          value: sonar
        - name: POSTGRES_USER
          value: sonar
        - name: POSTGRES_PASSWORD
          value: sonarpass
        ports:
        - containerPort: 5432
        volumeMounts:
        - name: postgres-data
          mountPath: /var/lib/postgresql/data
      volumes:
      - name: postgres-data
        emptyDir: {}
```

2) sonarqube-depolyment.yaml

파 일 이 름	sonarqube-deployment.yaml
파 일 설 명	sonarqube worker node 의 설정 파일입니다.
파 일 코 드	jenkins-deployment.yaml 파일 생성 후, 아래 코드 작성

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sonarqube
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sonarqube
  template:
    metadata:
      labels:
        app: sonarqube
    spec:
      nodeSelector:
        node-type: jenkins
        purpose: jenkins
      initContainers:
      - name: init-sysctl
        image: busybox:1.28
        command:
        - sysctl
        - -w
        - vm.max_map_count=262144
        securityContext:
          privileged: true
      containers:
      - name: sonarqube
        image: sonarqube:8.9.9-community
        ports:
        - containerPort: 9000
        env:
        - name: SONAR_JDBC_USERNAME
          value: sonar
        - name: SONAR_JDBC_PASSWORD
          value: sonarpass
        - name: SONAR_JDBC_URL
          value: jdbc:postgresql://sonar-postgres:5432/sonar
        volumeMounts:
        - name: sonar-data
          mountPath: /opt/sonarqube/data
        - name: sonar-extensions
```

```
          mountPath: /opt/sonarqube/extensions
      volumes:
      - name: sonar-data
        emptyDir: {}
      - name: sonar-extensions
        emptyDir: {}
```

3) sonarqube-service.yaml

파 일 이 름	sonarqube-service.yaml
파 일 설 명	sonarqube worker node 의 연동을 위한 파일입니다.
파 일 코 드	jenkins-deployment.yaml 파일 생성 후, 아래 코드 작성

```
apiVersion: v1
kind: Service
metadata:
  name: sonarqube
spec:
  type: NodePort
  ports:
  - port: 9000
    targetPort: 9000
    nodePort: 30900
  selector:
    app: sonarqube
---
apiVersion: v1
kind: Service
metadata:
  name: sonar-postgres
spec:
  ports:
  - port: 5432
  selector:
    app: sonar-postgres
```

6.3 SoarQube worker node 실행

Window PowerShell 관리자 권한 실행 후, 명령어 입력

1) 디플로이먼트와 서비스 배포

```
# SonarQube 폴더로 이동
$  cd C:\Users\Administrator\Desktop\DevSecOps\SonarQube

# deployment, service yaml 파일 배포
$  kubectl apply -f k8s\postgres-deployment.yaml
$  kubectl apply -f k8s\sonarqube-deployment.yaml
$  kubectl apply -f k8s\services.yaml

# 배포 상태 확인
$  kubectl get pods -o wide
$  kubectl get services
```

2) 웹 페이지 접속

```
Broser 주소창: localhost:30900 or http://Window Host IP:30900
```

3) localtunnel 을 이용한 도메인 연결

Window cmd 관리자 권한 실행

```
$  lt --subdomain sonarqube1234 --port 30900
```

7. Web 구축

7.1 Web 소스코드

방법 1) 직접 생성

1. DevSecOps 폴더에 Web 폴더 생성 후, 웹 소스코드 복사

   Practice Web Source code: DevSecOps Team Web Source code

2. Github 의 특수문자를 허용하기 위한 설정

   Window PowerShell 관리자 권한 실행 후, 명령어 입력

```
# DevSecOps\Web 폴더로 이동
$  cd C:\Users\Administrator\Desktop\DevSecOps\Web

# Github 특수문자 허용
$  git config core.protectNTFS false
```

방법 2) Window PowerShell 사용

Window PowerShell 관리자 권한 실행 후, 명령어 입력

```
# DevSecOps 폴더 이동
$  cd C:\Users\Administrator\Desktop\DevSecOps

# 소스코드 클론
$  git clone https://github.com/GH6679/web_wargamer.git
$  cd web_wargamer

# 깃허브 안에 있는 특수문자들 허용으로 설정
$  git config core.protectNTFS false
```

7.2 Web worker node 생성

DevSecOps 폴더에 Web 폴더 생성 후, yaml 설정 파일 생성

1) web-deployment.yaml 생성

파 일 이 름	web-deployment.yaml
파 일 설 명	Web worker node 의 설정 파일입니다.
파 일 코 드	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

파 일 이 름	db-deployment.yaml
파 일 설 명	웹용 데이터베이스 worker node 설정 파일입니다.
파 일 코 드	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-copnfigmap.yaml 생성

파일이름	db-init-configmap.yaml
파일설명	웹용 데이터베이스 worker node 의 초기화 설정 파일입니다.
파일코드	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) web-service.yaml 생성

파 일 이 름	web-service.yaml
파 일 설 명	웹과 데이터베이스의 연동을 위한 설정 파일입니다.
파 일 코 드	web-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

8. CI/CD 자동화 구축

8.1 Github Web hook 설정

Github Repository → Settings → Code and automation → Webhooks → Add webhook

- Payload URL: **https://jenkins1234.local.lt/github-webhook/** 입력
- Current type: **application/json** 선택
- SSL verification: **Enable SSL verification** 체크
- trigger: **Jest the push event.** 체크
- Active: **체크**

8.2 Github Access Token 생성

Github Profile → Settings → Developer settings → Personal access tokens → Tokens(classic) → Generate new token (classic)

- Note: **jenkin_token** 입력
- Select scopes**: repo, workflow, write:packages, admin:repo_hook** 체크

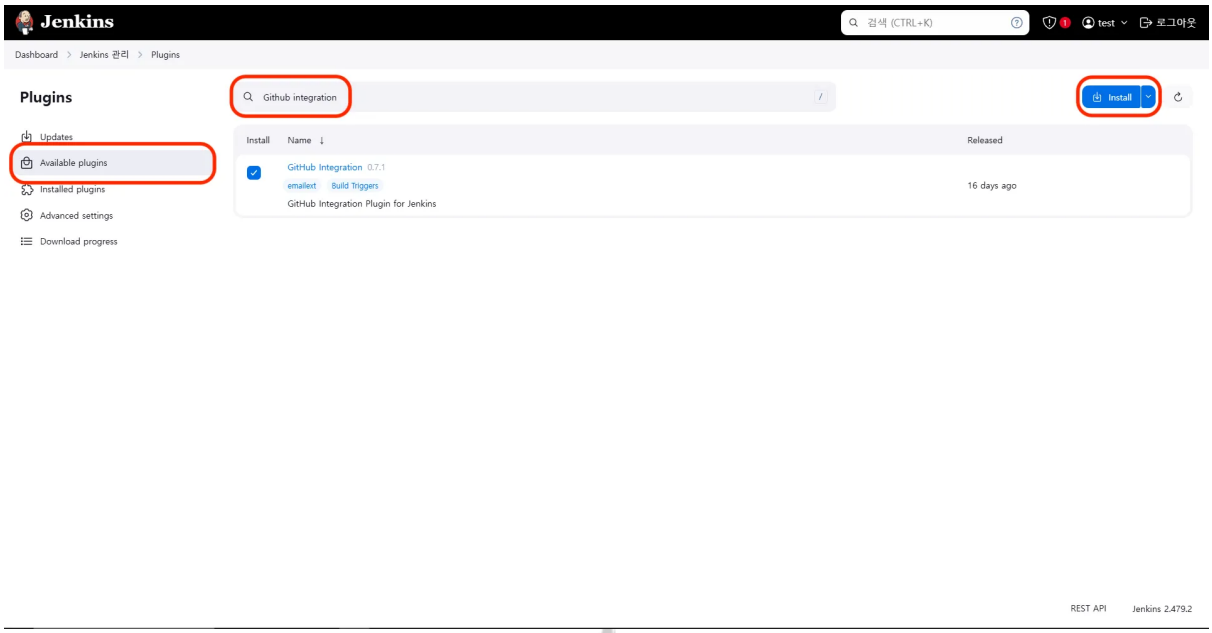
8.3 Jenkins/Github 연동

1) Jenkins 플러그인 설치

Jenkins 홈페이지 → Dashboard → Jenkins 관리 → Plugsins → Available plugins

Plugin	설명 및 목적
Github integration	Jenkins 와 Github 통합, 자동화된 빌드 및 PR 관리
SonarQube Scanner	코드 품질 및 보안 분석, SonarQube 와 통합
Sonar Quality Gates	품질 게이트 검증, 빌드 상태에 반영
Docker	Jenkins 에서 Docker 컨테이너 활용

Docker pipeline	Jenkins Pipeline 에서 Docker 통합



Jenkins plugin 설치 화면

2) Jenkins/Github 연동

Jenkins 홈페이지 → System → Global credentials → Add Credentials

- Username: **본인 Github Username** 입력
- Password: **8.2 에서 생성한 Github access token** 입력
- ID: **github_access_token** 입력

8.4 Jenkins/SonarQube 연동

1) Sonarqube Project 생성 및 token 생성

SonarQube 홈페이지 → Add a project → Manually → Project 생성 → 임의의 값 사용하여 token 생성

2) Jenkins Webhook/Sonarqube 연동

SonarQube 홈페이지 → Administration → configuration → Webhook → Create

- name: **jenkins-webhook** 입력
- URL: **https://jenkins1234.loca.lt/sonarqube-webhook/** 입력

3) Jenkins/Sonarqube 연동

Jenkins 홈페이지 → Dashboard → Jenkins 관리 → System

- **SonarQube server 설정**
  - Environment variables 체크
  - Name: **sonarqube-server** 입력
  - Server URL: **https://sonarqube1234.loca.lt** 입력
  - Server authentication token → Add 클릭하여 생성
    - Kind: **Secret text** 선택
    - Scope: **Global** 선택
    - Secret: **Sonarqube 에서 생성한 token 값** 입력
    - ID: **sonar-token** 입력

- **Github Servers 설정**
  - Name: **github-server** 입력
  - API URL: **https://api.github.com** 입력
  - Credentials → Add 클릭하여 생성
    - Kind: **Secret text** 선택
    - Scope: **Global** 선택
    - Secret: **Github access token 값** 입력
    - ID: **github-token** 입력

4) CI/CD 에서 사용될 도구 설정

Jenkins 홈페이지 → Dashbaord → Jenkins 관리 → Tools

Tool	목적	사용 예
Sonarqube Scanner	코드 품질 및 보안 분석 도구	빌드 완료 후 품질 분석 및 SonarQube 서버에 결과 업로드
Maven	프로젝트 빌드 및 의존성 관리	애플리케이션을 컴파일, 테스트, 패키징하여 JAR/WAR 파일 생성
Docker	컨테이너 기반 애플리케이션 빌드, 테스트, 배포	Jenkins 에서 Docker 이미지 빌드 및 컨테이너 실행. CI/CD 환경에서 배포 자동화

5) Jenkins Project 생성, Pipeline 스크립트 설정 및 테스트

Jenkins 홈페이지 → Dashboard → Create a job → Pipeline

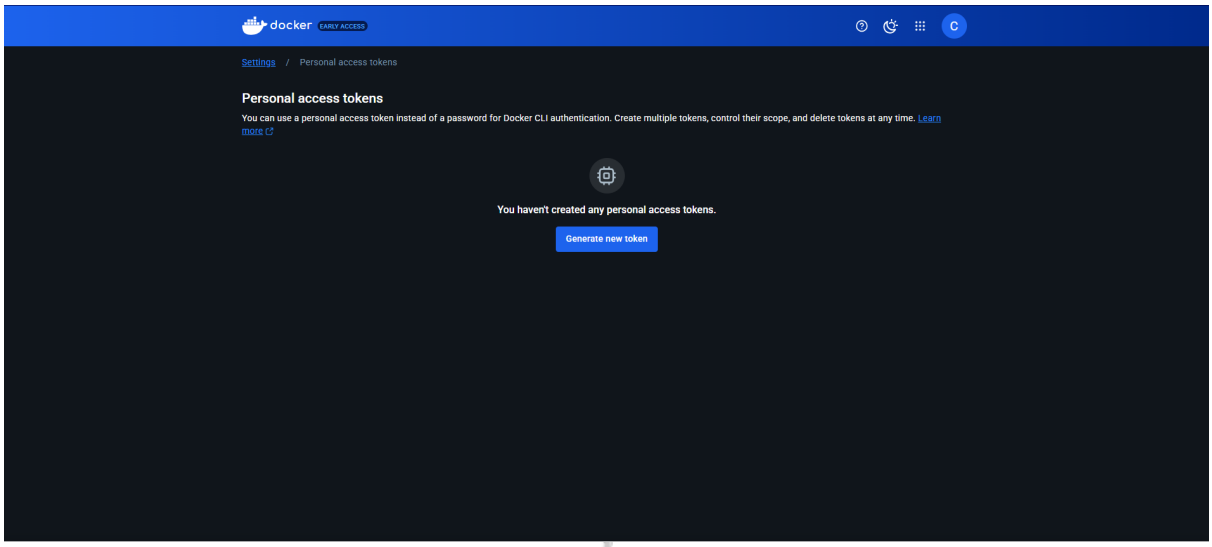
- Github Project: 체크
- Project url: Github Repository 에서 Clone using the web URL 후 붙여넣기
- Github hook trigger for GITScm polling: 체크
- Pipeline script 입력: **url 주소에 Project url 과 동일한 URL 입력**

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git branch: 'main',
                    credentialsId: 'github_access_token',
                    url: 'https://github.com/nickName/reponame.git'
            }
        }
    }
}
```

- 지금 빌드 클릭하여 Jenkins 와 Github 연동 테스트

6) Docker Access token 생성

Docker Desktop → Profile → Account setting → Personal Access token → Generate new token



7) Jenkins worker node 에 Docker 설치 및 레지스터 설정

Window PowerShell 관리자 권한 실행 후, 다음 코드 입력

```
# pod 상태 확인
$  kubectl get pods -o wide

# jenkins IP 확인
```

```
PS C:\Users\Junseong\Desktop\devsecops_full_pro-main\Jenkins> kubectl get pods -o wide
NAME                              READY   STATUS    RESTARTS   AGE    IP           NODE
     NOMINATED NODE   READINESS GATES
jenkins-6b68fb88c9-rg5bb          1/1     Running   0          106m   10.244.4.2   devsecops-cluster-worke
r2   <none>           <none>
sonar-postgres-6d4647bdc6-ffkpb   1/1     Running   0          24m    10.244.4.4   devsecops-cluster-worke
r2   <none>           <none>
sonarqube-5977d4c64f-bpf4n        1/1     Running   0          24m    10.244.4.3   devsecops-cluster-worke
r2   <none>           <none>
```

```
# Jenkins worker node 접속
$  kubectl exec -it [jenkins worker node ID] -- /bin/bash

# Docker 설치에 필요한 패키지 설치
$  apt-get update
$  apt-get install -y apt-transport-https ca-certificates curl gnupg
   lasb-release
```

```
# Docker GPG 키 추가
$  curl  -fsSL  https://download.docker.com/linux/debian/gpg  |  gpg  --
   dearmor -o /etc/apt/keyrings/docker.헿

# Debian Docker 저장소 추가
$  echo     "deb     [arch=$(dpkg     --print-architecture)     signed-
   by=/etc/apt/keyrings/docker.gpg]
   https://download.docker.com/linux/debian  bookworm  stable"  |  tee
   /etc/apt/sources.list.d/docker.list > /dev/null

# 패키지 목록 업데이트 및 Docker 설치
$  apt-get update
$  apt-get install -y docker-ce docker-ce-cli containerd.io

# docker 파일 수정
$  apt-get install -y vim
$  vi /etc/init.d/docker
  # 62 번째 줄을 찾아 다음과 같이 수정
  # 기존: ulimit -Hn 524288
  # 수정: ulimit -n 524288

# Docker 실행 및 docker.sock 생성 확인
$  ls -l /var/run/docker.sock

# 레지스트리를 5000 포트로 실행
$  docker run -d -p 5000:5000 --restart=always --name registry registry:2

# 레지스트리 설정
$  mkdir -p /etc/docker
$  echo '{
     "insecure=registries": ["[Jenkins IP]:5000"],
     "storage-driver": "vfs"
   } > /etc/docker/daemon.json

# 레지스트리 적용 여부 확인
$  service docker restart
```

```
$  curl http://[Jenkins IP]:5000/v2/_catalog

# Docker 소켓 권한 설정
$  chmod 666 /var/run/docker.sock

# Docker Option 설정 파일 생성
$  mkdir -p /etc/systemd/system/docker.service.d
$  vi /etc/systemd/system/docker.service.d/docker-options.conf

# docker-options.conf 파일에 아래 내용 추가
--insecure-registry=[Jenkins IP]:5000
```

8) Web worker node 에 Docker 설치 및 레지스터 설정

Window PowerShell 관리자 모드 실행 후, 다음 코드 입력

```
# Web worker node 접속
$  kubectl debug node/devsecops-cluster-worker -it --image=ubuntu --
   chroot /host bash

# Daemon 설정
$  mkdir -p /etc/docker
$  echo '{
     "insecure-registries": ["[Jenkins IP]:5000"],
     "storage=driver" "vfs"
   }' > /etc/docker/daemon.json

# kubelet 설정
$  mkdir -p /etc/kubernetes/registies
$  echo '{
     "apiVersion": "kubelet.config.k8s.io/v1",
     "kind": "CredentialProviderConfig",
     "providers": [{
       "name": "registry-credentials",
       "matchImages": ["[Jenkins IP]:5000/*"],
       "defaultCacheDuration": "12h"
     }]
   }' > /etc/kubernetes/registries/config.json
```

```
# 네트워크 설정
$  echo  “NO_PROXY=10.0.0.0/8,  172.16.0.0/12,  192.168.0.0/16,  svc,
   localhost, 127.0.0.1” >> /etc/environment
$  echo “no_proxy=$NO_PROXY” >> /etc/environment

# 레지스트리 endpoint 설정
$  mkdir -p /etc/containerd
$  cat > /etc.containerd/config.toml << EOF
   version = 2
   [plugins.”io.containerd.grpc.v1.cri”.registry]
     [plugins.”io.containerd.grpc.v1.cri”.registry.mirrors]

   [plugins.”io.containerd.grpc.v1.cri”.registry.mirrors.”[Jenkin
   IP]:5000”]
         endpoint = [“http://[Jenkins IP]:5000]]
     [plugins.”io.containerd.grpc.v1.cri”.registry.configs]

   [plugins.”io.containerd.grpc.v1.cri”.registry.configs.”[Jenkins
   IP]:5000”.tls]
         insecure_skip_verify = true
   EOF

# Containerd 재시작 및 설정 적용 후 레지스트리 확인
$  systemctl restart containerd
$  kubectl  debug  node/devsecops-cluster-worker  -it  --image=ubuntu  –
   chroot /host bash
$  curl http://[Jenkins IP]:5000/v2/_catalog
```

9) CI/CD Pipeline script 설정

Jenkins 홈페이지 → Dashboard → 8.4-5)에서 생성한 Pipeline script 수정

※ **[url]**: 본인의 Github Repository URL 입력

※ **[Jenkins IP]**: Jenkins worker node IP 입력

※ **[Docker 계정 이름]**: Docker Desktop 계정의 이름 입력

※ **[Docker 토큰 값]**: 8.4-6)에서 얻은 Docker 토큰

※ **[Maven 값]**: 8.4-1)에서 얻은 Maven 값

※ **[Host IP]**: 본인의 localhost IP 또는 127.0.0.1

```
pipeline {
    agent any
    tools {
        maven 'Maven'
    }

    triggers {
        githubPush()
    }

    stages {
        stage('Checkout') {
            steps {
                git branch: 'main',
                    credentialsId: 'github_access_token',
                    url: ['https://github.com/nickName/reponame.git']
            }
        }

        stage('Install Dependencies') {
            steps {
                sh '''
                    curl  -LO  "https://dl.k8s.io/release/$(curl  -L  -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
                    chmod +x kubectl
                    mv kubectl /usr/local/bin/

                    # Docker 데몬 설정
                    mkdir -p /etc/docker
                                    echo '{
                      "insecure-registries": ["[Jenkins IP]:5000"],
                      "storage-driver": "vfs"
                    }' > /etc/docker/daemon.json

                    # Docker 소켓 권한 설정
                    chmod 666 /var/run/docker.sock
                '''
            }
        }

        stage('SonarQube analysis') {
```

```
            steps {
                dir('web_wargamer-master') {
                    withSonarQubeEnv('sonarqube-server') {
                        sh "mvn sonar:sonar \
                            -Dsonar.projectKey=test \
                            -Dsonar.host.url=http://[Host IP]:30900/ \
                            -Dsonar.login=[Maven 값]
                            -Dsonar.sources=. \
                            -Dsonar.java.binaries=."
                    }
                }
            }
        }

        stage('Build Docker Image') {
            steps {
                dir('web_wargamer-master') {
                    sh """
                        # Docker 데몬 상태 확인
                        if ! docker info > /dev/null 2>&1; then
                            dockerd &
                            sleep 5
                        fi

                        # 이미지 빌드
                        docker build -t wargame-web:latest .

                        # 이미지 태그 및 푸시
                        # 본인의 Jenkins IP 입력 (포트 변경 X)
                        docker tag wargame-web:latest [Jenkins
IP]:5000/wargame-web:latest
                        docker push [Jenkins IP]:5000/wargame-web:latest
                    """
                }
            }
        }

        stage('Create Registry Secret') {
            steps {
                sh """
```

```
                # docker-username 에는 자기 docker 계정이름 적기
                # docker-password 는 위에서 얻은 docker token 넣기
                    kubectl create secret docker-registry registry-secret
\
                      --docker-server=[Jenkins IP]:5000 \
                      --docker-username=[Docker 계정 이름] \
                      --docker-password=[Docker 토큰 값] \
                      --dry-run=client -o yaml | kubectl apply -f -
                """
            }
        }
                        stage('Deploy to Kubernetes') {
                            steps {
                                script {
                                    // 기존 설정 적용
                                    sh """
                                        kubectl  apply  -f  web_wargamer-
master/k8s/db-init-configmap.yaml
                                        kubectl  apply  -f  web_wargamer-
master/k8s/db-deployment.yaml
                                        kubectl  apply  -f  web_wargamer-
master/k8s/web-deployment.yaml
                                        kubectl  apply  -f  web_wargamer-
master/k8s/services.yaml
                                    """

                                    // Deployment 롤링 업데이트 강제 실행
                                    sh """
                                        kubectl rollout restart deployment
wargame-web
                                        kubectl rollout status deployment
wargame-web
                                    """
                                }
                            }
                        }

    }
}
```

TEAM-DEVSECOPS