

Homework Sesi 31 Bootcamp DevOps Engineer Digital Skola

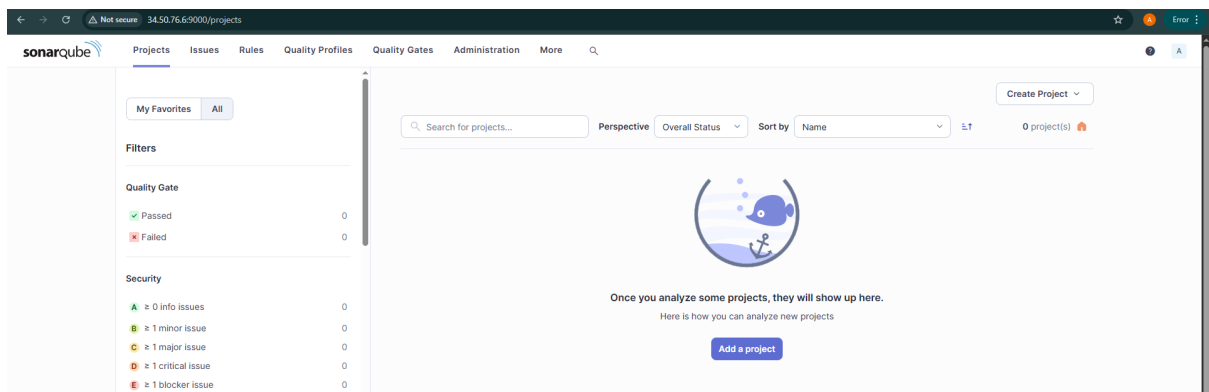
Buat 1 pipeline cicd yang terintegrasi dengan sonarqube dan jelaskan step by stepnya

Dalam project ini, pipeline CI/CD yang telah dibuat di project sebelumnya dimodifikasi dengan menambahkan SonarQube.

SonarQube akan menganalisis kode sumber (dalam kasus ini, index.html dan Dockerfile) untuk potensi bug, kerentanan, dan code smells *sebelum* image Docker dibangun.

Prasyarat:

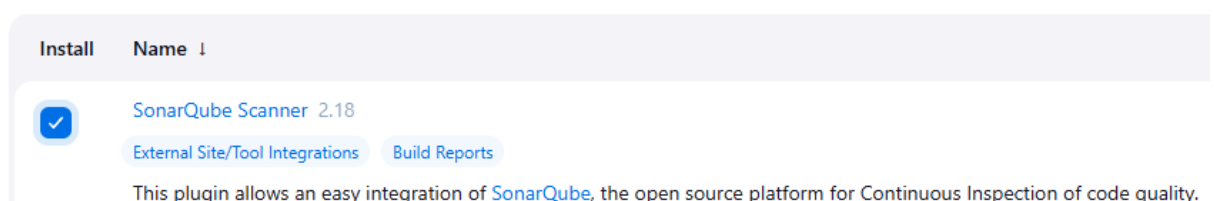
1. **SonarQube Server:** Harus ada instance SonarQube yang berjalan dan dapat diakses dari Jenkins server/agent.



2. **SonarScanner:** Alat command-line SonarScanner perlu tersedia untuk Jenkins. Cara termudah adalah mengkonfigurasinya melalui Jenkins Global Tool Configuration.
3. **Jenkins SonarQube Scanner Plugin:** Plugin SonarQube Scanner for Jenkins perlu diinstall di Jenkins (Manage Jenkins -> Plugins -> Available -> cari "SonarQube Scanner" -> Install).

Langkah 1: Konfigurasi SonarQube di Jenkins

1. **Install Plugin:** Pastikan plugin SonarQube Scanner for Jenkins sudah terinstall.



2. **Generate SonarQube Token:**

- Buka SonarQube UI.
- Login sebagai admin atau user dengan hak yang cukup.

- Navigasi ke Administration -> Security -> Users.
- Klik ikon token untuk user yang akan digunakan Jenkins (atau buat user teknis khusus).
- Generate token baru (misalnya, beri nama jenkins-token).
- **Salin token ini dengan aman.** Token tidak akan bisa dilihat lagi.

3. Tambahkan SonarQube Token ke Jenkins Credentials:

- Di Jenkins: Manage Jenkins -> Credentials -> (global) -> Add Credentials.
- Kind: Secret text.
- Secret: Tempelkan token SonarQube yang baru saja dibuat.
- ID: sonarqube-token (atau ID deskriptif lainnya).
- Description: (Opsional) Token untuk otentikasi SonarQube.
- Klik Create.

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain

Global credentials (unrestricted) ▼

Kind

Secret text ▼

Scope ?

Global (Jenkins, nodes, items, all child items, etc) ▼

Secret

.....

ID ?

sonarqube-auth-token

Description ?

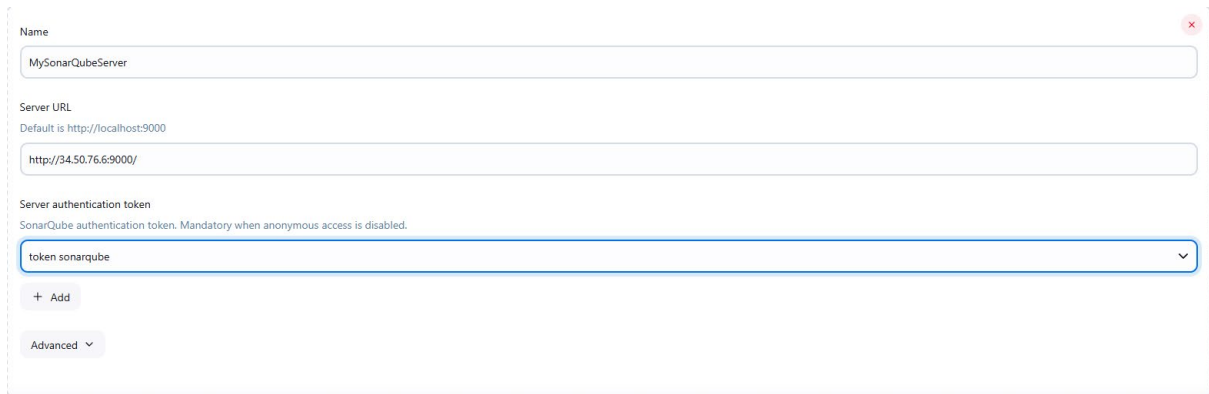
token sonarqube

Cancel Add

4. Konfigurasi SonarQube Server:

- Di Jenkins: Manage Jenkins -> Configure System.
- Scroll ke bagian SonarQube servers.
- Klik Add SonarQube.
- Name: Beri nama yang mudah dikenali (misal: MySonarQubeServer). Nama ini akan digunakan di Jenkinsfile.

- Server URL: Masukkan URL instance SonarQube (misal: `http://sonar.yourdomain.com` atau `http://<SONARQUBE_IP>:9000`).
- Server authentication token: Pilih ID kredensial 'Secret text' yang baru saja dibuat (sonarqube-token).
- Klik Save.

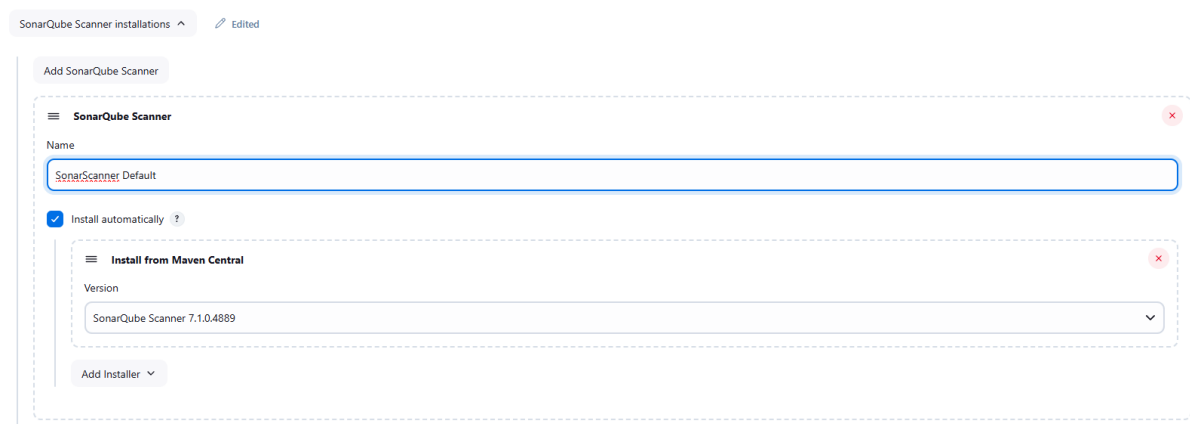


The screenshot shows a configuration form for a SonarQube server. It includes fields for 'Name' (MySonarQubeServer), 'Server URL' (http://34.50.76.6:9000/), and 'Server authentication token' (token sonarqube). There is an 'Add' button and an 'Advanced' dropdown menu.

5. Konfigurasi SonarScanner Tool:

- Di Jenkins: Manage Jenkins -> Global Tool Configuration.
- Scroll ke bagian SonarQube Scanner.
- Klik Add SonarQube Scanner.
- Name: Beri nama (misal: SonarScanner Default).
- Centang Install automatically (cara termudah). Jenkins akan mengunduh versi terbaru saat dibutuhkan. Atau, jika sudah terinstall manual di Jenkins agent/server, uncheck dan berikan path instalasi (SONAR_SCANNER_HOME).
- Klik Save.

SonarQube Scanner installations



The screenshot shows the 'SonarQube Scanner installations' page in Jenkins. It includes a 'SonarQube Scanner' configuration card with a 'Name' field (SonarScanner Default), an 'Install automatically' checkbox (checked), and an 'Install from Maven Central' section with a 'Version' dropdown (SonarQube Scanner 7.1.0.4889). There is an 'Add Installer' button.

Langkah 2: Membuat Project di SonarQube (Opsional)

SonarQube dapat secara otomatis membuat project saat analisis pertama kali dijalankan, atau dapat dibuat secara manual terlebih dahulu:

- **Manual:** Di SonarQube UI, klik Create Project, pilih Manually. Berikan Project key (misal: simple-nginx-app) dan Display name.
- **Otomatis:** Pastikan user yang digunakan oleh Jenkins (melalui token) memiliki izin 'Create Projects'. Project akan dibuat dengan Project key yang didefinisikan di Jenkinsfile.

Langkah 3: Memodifikasi Jenkinsfile

Tambahkan stage baru untuk analisis SonarQube, idealnya setelah checkout kode dan sebelum build image.

```
// Mendefinisikan variabel global yang akan digunakan di seluruh pipeline
def DOCKERHUB_CREDENTIALS_ID = 'dockerhub-credentials' // Sesuaikan dengan ID
kredensial Docker Hub di Jenkins
def TARGET_SERVER_SSH_CREDENTIALS_ID = 'web-app-server-ssh-key' // Sesuaikan
dengan ID kredensial SSH di Jenkins
def GITHUB_REPO_NAME = 'simple-nginx-app' // Sesuaikan dengan nama repo
(digunakan untuk nama image)
def DOCKERHUB_USERNAME = 'avimajid' // Ganti dengan username Docker Hub
def TARGET_SERVER_IP = '34.101.223.209' // Ganti dengan IP atau hostname
server target
def TARGET_SERVER_USER = 'userdeploy' // Ganti dengan user SSH di server
target
def CONTAINER_NAME = 'my-nginx-app' // Nama kontainer yang akan berjalan di
server target
def APP_PORT = 9090 // Port di host target yang akan di-map ke port 80
kontainer

// Variabel Spesifik SonarQube
def SONARQUBE_SERVER_NAME = 'MySonarQubeServer' // Sesuaikan dengan Nama
Server SonarQube di Jenkins Config
def SONARQUBE_PROJECT_KEY = "com.yomamen:${GITHUB_REPO_NAME}" // Kunci Unik
Project di SonarQube (sesuaikan!)

pipeline {
    agent any // Menjalankan pipeline di agent Jenkins mana saja yang tersedia

    environment {
        // Membuat nama image unik dengan nomor build Jenkins
```

```

                                IMAGE_NAME           =
"${DOCKERHUB_USERNAME}/${GITHUB_REPO_NAME}:${env.BUILD_NUMBER}"
                                LATEST_IMAGE_NAME      =
"${DOCKERHUB_USERNAME}/${GITHUB_REPO_NAME}:latest"
    // Path ke Trivy (sesuaikan jika perlu)
    TRIVY_PATH = '/snap/bin/trivy' // Ganti jika path instalasi Trivy
berbeda
}

stages {
    stage('1. Checkout Code') {
        steps {
            echo 'Mengambil kode sumber dari GitHub...'
            // Mengambil kode dari SCM (GitHub) yang dikonfigurasi di job
Jenkins

            checkout scm
        }
    }
    stage('2. SonarQube Analysis') {
        steps {
            script {
                // Menggunakan nama Konfigurasi SonarQube Server dari
Jenkins System Config
                // dan nama Konfigurasi SonarScanner dari Jenkins Global
Tools Config

                def scannerHome = tool name: 'SonarScanner Default', type:
'hudson.plugins.sonar.SonarRunnerInstallation'
                // Menggunakan wrapper untuk inject URL & Token SonarQube
                withSonarQubeEnv(SONARQUBE_SERVER_NAME) {
                    sh """
                        ${scannerHome}/bin/sonar-scanner \
                        -Dsonar.projectKey=${SONARQUBE_PROJECT_KEY} \
                        -Dsonar.sources=. \
                        -Dsonar.projectName=${GITHUB_REPO_NAME} \
                        -Dsonar.projectVersion=${env.BUILD_NUMBER} \
                        -Dsonar.host.url=${env.SONAR_HOST_URL} \
                        -Dsonar.login=${env.SONAR_AUTH_TOKEN}
                    """
                    // Catatan: -Dsonar.host.url dan -Dsonar.login di-
inject oleh withSonarQubeEnv
                    // Jika ada file konfigurasi sonar-project.properties
di repo, beberapa -D flag bisa dihilangkan
                }
            }
        }
    }
}

stage('2. Build Docker Image') {

```

```

        steps {
            script {
                echo "Membangun image Docker: ${IMAGE_NAME}"
                // Menggunakan plugin Docker Pipeline untuk build image
                docker.build(IMAGE_NAME, '.') // '.' berarti Dockerfile
            }
        }
    }

    stage('3. Scan Image with Trivy') {
        steps {
            echo "Memindai image ${IMAGE_NAME} dengan Trivy..."
            // Menjalankan Trivy dari shell. Pipeline akan gagal jika
            // ditemukan kerentanan HIGH atau CRITICAL (--exit-code 1)
            // --ignore-unfixed digunakan agar tidak gagal karena vuln yg
            // belum ada patchnya
            // Sesuaikan severity sesuai kebutuhan (misal: 'CRITICAL'
            // saja)
            sh "trivy image --exit-code 1 --severity HIGH,CRITICAL --
            ignore-unfixed ${IMAGE_NAME}"
        }
    }

    stage('4. Push Image to Docker Hub') {
        // Stage ini hanya berjalan jika stage sebelumnya (Scan) berhasil
        steps {
            script {
                echo "Mendorong image ${IMAGE_NAME} ke Docker Hub..."
                // Menggunakan Docker Hub credentials yang sudah
                // dikonfigurasi
                docker.withRegistry('https://registry.hub.docker.com',
                DOCKERHUB_CREDENTIALS_ID) {
                    // Push image dengan tag nomor build
                    docker.image(IMAGE_NAME).push()
                    // Juga push image dengan tag 'latest'
                    docker.image(IMAGE_NAME).push('latest')
                    echo "Image ${IMAGE_NAME} dan ${LATEST_IMAGE_NAME}
                    berhasil didorong."
                }
            }
        }
    }

    stage('5. Deploy to Target Server') {
        // Stage ini hanya berjalan jika stage sebelumnya (Push) berhasil
        steps {

```

```

        echo "Mendeploy image ${LATEST_IMAGE_NAME} ke server target
${TARGET_SERVER_IP}..."
        // Menggunakan SSH Agent plugin dengan kredensial SSH yang
dikonfigurasi
        sshagent([TARGET_SERVER_SSH_CREDENTIALS_ID]) {
            // Menjalankan perintah di server target via SSH
            sh """
                ssh -o StrictHostKeyChecking=no
${TARGET_SERVER_USER}@${TARGET_SERVER_IP}
                echo 'Menarik image terbaru dari Docker Hub...'
                docker pull ${LATEST_IMAGE_NAME}

                echo 'Menghentikan dan menghapus kontainer lama
(jika ada)...'

                docker stop ${CONTAINER_NAME} || true
                docker rm ${CONTAINER_NAME} || true

                echo 'Menjalankan kontainer baru...'
                docker run -d --name ${CONTAINER_NAME} -p
${APP_PORT}:80 ${LATEST_IMAGE_NAME}

                echo 'Deployment selesai. Aplikasi berjalan di
http://${TARGET_SERVER_IP}:${APP_PORT}'
                """
        }
    }
}

post {
    // Tindakan yang dilakukan setelah semua stage selesai, terlepas dari
statusnya
    always {
        echo 'Pipeline selesai.'
        // Membersihkan workspace Jenkins
        cleanWs()
    }
    success {
        echo 'Pipeline berhasil!'
        // Bisa ditambahkan notifikasi (email, Slack, dll.)
    }
    failure {
        echo 'Pipeline gagal!'
        // Bisa ditambahkan notifikasi kegagalan
    }
    unstable {
        // Biasanya karena test gagal, tapi tidak relevan di pipeline ini
        echo 'Pipeline tidak stabil.'
    }
}

```

```
}  
}  
}
```

Penjelasan stage sonarqube Jenkinsfile:

1. **Variabel Baru:** SONARQUBE_SERVER_NAME (harus cocok dengan nama di Konfigurasi Jenkins) dan SONARQUBE_PROJECT_KEY (kunci unik untuk project ini di SonarQube).
2. **Stage 'SonarQube Analysis':**
 - tool name: 'SonarScanner Default': Mengambil path instalasi SonarScanner yang dikonfigurasi di Global Tool Configuration Jenkins. Sesuaikan namanya jika berbeda.
 - withSonarQubeEnv(SONARQUBE_SERVER_NAME): Wrapper dari plugin SonarQube Scanner. Ini secara otomatis:
 - Mengambil URL server dan token otentikasi dari konfigurasi Jenkins.
 - Menyediakannya sebagai environment variables (SONAR_HOST_URL, SONAR_AUTH_TOKEN) di dalam bloknya.
 - sh "\${scannerHome}/bin/sonar-scanner ...": Menjalankan SonarScanner.
 - -Dsonar.projectKey: Mengidentifikasi project di SonarQube. **Penting untuk dibuat unik.**
 - -Dsonar.sources=.: Menentukan bahwa kode sumber yang akan dianalisis ada di direktori kerja saat ini (root workspace).
 - -Dsonar.projectName: Nama yang akan ditampilkan di UI SonarQube.
 - -Dsonar.projectVersion: Versi analisis (menggunakan nomor build Jenkins).
 - -Dsonar.host.url dan -Dsonar.login: Diambil otomatis dari withSonarQubeEnv.
3. **Stage 'Check Quality Gate':**
 - waitForQualityGate abortPipeline: true: Langkah dari plugin SonarQube Scanner yang akan menunggu SonarQube selesai memproses laporan analisis.
 - abortPipeline: true: Jika Quality Gate di SonarQube berstatus ERROR (gagal), langkah ini akan menghentikan (abort) pipeline Jenkins.
 - timeout: 5: Memberi batas waktu (dalam menit) untuk menunggu hasil dari SonarQube.

Langkah 4: Jalankan dan Verifikasi

1. Commit dan push perubahan Jenkinsfile ke GitHub.
2. Jalankan pipeline nginx-cicd-pipeline di Jenkins.
3. Perhatikan stage baru "SonarQube Analysis".
4. Periksa Console Output untuk log dari SonarScanner. Seharusnya terlihat koneksi ke SonarQube dan analisis kode.
5. Jika berhasil, periksa project di SonarQube UI. Seharusnya terlihat hasil analisis untuk build tersebut.
6. Periksa status Quality Gate di SonarQube dan pastikan pipeline Jenkins bereaksi sesuai (berlanjut jika OK/WARN, berhenti jika ERROR).
7. **Catatan:** Untuk project Nginx sederhana ini (hanya HTML dan Dockerfile), SonarQube mungkin tidak melaporkan banyak isu kecuali jika dikonfigurasi secara spesifik untuk menganalisis Dockerfile atau aturan HTML/CSS yang ketat diaktifkan. Manfaatnya akan lebih terasa pada project dengan bahasa pemrograman seperti Java, Python, JavaScript, dll.

Sekarang pipeline telah dilengkapi dengan analisis kualitas kode statis menggunakan SonarQube sebelum melanjutkan ke proses build dan deployment.

Verifikasi:

Setelah beberapa penyesuaian dan troubleshooting, akhirnya pipeline CI/CD berhasil di build

The screenshot shows the Jenkins pipeline execution status for build #27. At the top, there is a green checkmark icon and the text "#27". Below this, it says "Success 1 min 18 sec ago in 1 min 11 sec". On the left, a list of stages is shown with green checkmarks: "Checkout SCM", "1. Checkout Code", "2. SonarQube Analysis", "3. Build Docker Image", "4. Scan Image with Trivy", "5. Push Image to Docker Hub", "6. Deploy to Target Server", and "Post Actions". On the right, a detailed view of the "Post Actions" stage is shown, containing three items: "Pipeline selesai." with a right arrow, "Delete workspace when build is done" with a right arrow, and "Pipeline berhasil!" with a dropdown arrow. Below these items, a message "Pipeline berhasil!" is displayed with a blue icon.