Homework Sesi 29 Bootcamp DevOps Engineer Digital Skola

Buat 1 proses cicd yang menerapkan proses devsecops didalamnya, dan jelaskan step by step

Prasyarat:

1. **Server Jenkins:** Jenkins sudah terinstal dan berjalan. Akses ke antarmuka web Jenkins diperlukan.
2. **Docker di Server Jenkins:** Docker harus terinstal dan service Docker berjalan di server tempat Jenkins diinstal. Pengguna Jenkins (jenkins atau pengguna yang menjalankan proses Jenkins) harus memiliki izin untuk menjalankan perintah Docker (biasanya ditambahkan ke grup docker).
3. **Trivy di Server Jenkins:** Trivy harus sudah terinstal di server Jenkins dan dapat diakses melalui command line.
4. **Server Target:** Server Linux terpisah sebagai tujuan deployment. Server ini harus memiliki Docker terinstal dan dapat diakses melalui SSH dari server Jenkins.
5. **Akun GitHub:** Akun GitHub untuk menyimpan kode sumber aplikasi.
6. **Repositori GitHub:** Repositori yang berisi kode sumber aplikasi web Nginx sederhana dan Dockerfile.
7. **Plugin Jenkins:** Plugin berikut perlu diinstal di Jenkins (melalui Manage Jenkins -> Plugins):
    - Pipeline (biasanya sudah terinstal)
    - Git (biasanya sudah terinstal)
    - Docker Pipeline
    - SSH Agent (untuk mengelola kunci SSH saat deployment)

Langkah 1: Persiapan Repositori GitHub

1. **Buat Repositori:** Buat repositori baru di GitHub (misalnya, simple-nginx-app).
2. **Buat File Aplikasi:** Buat file index.html sederhana di direktori root repositori:

```
<!DOCTYPE html>
<html>
<head>
    <title>Selamat Datang di Nginx!</title>
```

```
</head>
<body>
    <h1>Halo dari Nginx via GitLab CI/CD!</h1>
    <p>Aplikasi ini dideploy secara otomatis.</p>
</body>
</html>
```

3. **Buat Dockerfile:** Buat file bernama Dockerfile (tanpa ekstensi) di direktori root repositori:

```
# Gunakan base image Nginx resmi yang ringan
FROM nginx:alpine

# Salin file index.html ke direktori default Nginx
COPY index.html /usr/share/nginx/html/index.html

# Ekspos port 80
EXPOSE 80

# Perintah default untuk menjalankan Nginx saat container dimulai
CMD ["nginx", "-g", "daemon off;"]
```

Langkah 2: Konfigurasi Jenkins

1. **Konfigurasi Kredensial GitHub:**
    - Untuk mengizinkan Jenkins mengakses repositori privat (atau untuk menghindari limit rate pada repositori publik), disarankan menggunakan **Personal Access Token** (**PAT**) GitHub.
    - Buat PAT di GitHub (Settings -> Developer settings -> Personal access tokens -> Tokens (classic) -> Generate new token). Berikan scope repo. Salin token yang dihasilkan.
    - Di Jenkins, buka Manage Jenkins -> Credentials -> System -> Global credentials (unrestricted).
    - Klik Add Credentials.
    - Pilih Kind: Secret text.
    - Masukkan token GitHub ke kolom Secret.
    - Berikan ID yang mudah diingat (misalnya, **github-token-ci**). ID ini akan digunakan di Jenkinsfile.
    - Klik Create.

2. **Konfigurasi Kredensial SSH untuk Server Target:**
    - Agar Jenkins dapat mendeploy ke server target via SSH, perlu disiapkan pasangan kunci SSH.
    - Di server Jenkins (atau mesin lain), generate pasangan kunci SSH jika belum ada: **ssh-keygen -t rsa -b 4096**.
    - Salin *public key* (~/.ssh/id_rsa.pub) ke file ~/.ssh/authorized_keys di server target pada user yang akan digunakan untuk deployment (misalnya, userdeploy). Pastikan permissions file dan direktori ~/.ssh benar (700 untuk direktori, 600 untuk authorized_keys).
    - Di Jenkins, buka Manage Jenkins -> Credentials -> System -> Global credentials (unrestricted).
    - Klik Add Credentials.
    - Pilih Kind: SSH Username with private key.
    - Berikan ID yang mudah diingat (misalnya, **web-app-server-ssh-key**).
    - Masukkan Username yang digunakan di server target (misalnya, userdeploy).
    - Pada bagian Private Key, pilih Enter directly dan tempelkan *private key* (~/.ssh/id_rsa).
    - Klik Create.
3. **Konfigurasi Kredensial Docker Hub:**
    - Di halaman "Credentials" yang sama, klik "Add Credentials" lagi.
    - Pilih Kind: "Username with password".
    - Scope: "Global".
    - ID: Berikan ID unik (misalnya, **dockerhub-credentials**). ID ini akan digunakan di Jenkinsfile.
    - Description: Deskripsi singkat.
    - Username: Username Docker Hub.
    - Password: Password Docker Hub atau Access Token (lebih disarankan).
    - Klik "OK".Masukkan Username yang digunakan di server target (misalnya, userdeploy).
    - Pada bagian Private Key, pilih Enter directly dan tempelkan *private key* (~/.ssh/id_rsa).

Langkah 3: Membuat Jenkins Pipeline Job

1. Di dashboard Jenkins, klik New Item.
2. Masukkan nama item (misalnya, **nginx-app-cicd**).
3. Pilih Pipeline dan klik OK.
4. Pada halaman konfigurasi job:
    - Berikan deskripsi untuk pipeline (opsional).
    - Gulir ke bawah ke bagian Pipeline.
    - Pilih Definition: Pipeline script from SCM.
    - Pilih SCM: Git.
    - Masukkan Repository URL repositori GitHub (https://github.com/avikaaffah/simple-nginx-app.git).
    - Pilih Credentials yang sudah dibuat untuk GitHub (misalnya, **github-token-ci**). Jika repositori publik, bisa pilih None.
    - Pastikan Branch Specifier sesuai dengan branch utama (misalnya, */main atau */master).
    - Biarkan Script Path sebagai Jenkinsfile (nama default).
5. Klik Save.

Langkah 4: Menulis Jenkinsfile

Sekarang, buat file bernama Jenkinsfile di root direktori repositori GitHub. File ini mendefinisikan tahapan pipeline.

```
// 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

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

        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.'
            }
        }
}
```

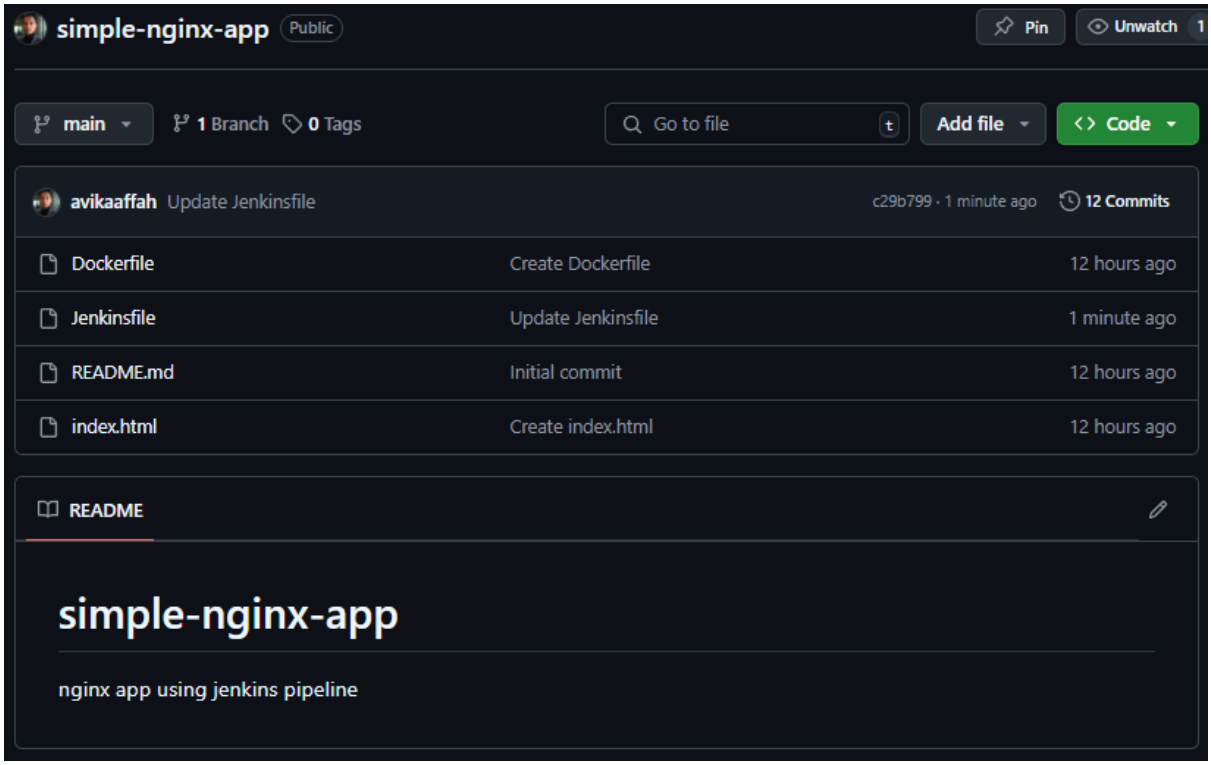
Penjelasan Jenkinsfile:

- **def ...**: Mendefinisikan variabel untuk kredensial, nama image, server target, dll. agar mudah diubah. **PENTING:** Ganti nilai variabel DOCKERHUB_USERNAME, TARGET_SERVER_IP, TARGET_SERVER_USER, dan APP_PORT sesuai dengan konfigurasi spesifik. Sesuaikan juga ID kredensial jika berbeda.
- **pipeline { ... }**: Blok utama yang mendefinisikan pipeline deklaratif.
- **agent any**: Menentukan bahwa pipeline dapat berjalan di node Jenkins mana pun yang tersedia.

- **environment { ... }**: Mendefinisikan variabel lingkungan yang tersedia di semua stage. IMAGE_NAME dibuat unik menggunakan variabel lingkungan env.BUILD_NUMBER yang disediakan Jenkins.
- **stages { ... }**: Berisi urutan tahapan (stages) yang akan dieksekusi.
- **stage('...') { ... }**: Mendefinisikan satu tahapan dalam pipeline.
    - **Checkout Code**: Menggunakan checkout scm untuk mengambil kode dari Git.
    - **Build Docker Image**: Menggunakan docker.build() dari plugin Docker Pipeline untuk membuat image.
    - **Scan Image with Trivy**: Menjalankan perintah trivy dalam sh step. --exit-code 1 membuat stage gagal jika ditemukan kerentanan dengan tingkat keparahan (--severity) HIGH atau CRITICAL. --ignore-unfixed mencegah kegagalan karena kerentanan yang belum memiliki perbaikan.
    - **Push Image to Docker Hub**: Menggunakan docker.withRegistry() untuk autentikasi ke Docker Hub menggunakan kredensial yang disimpan di Jenkins, lalu image.push() untuk mendorong image dengan tag nomor build dan tag latest.
    - **Deploy to Target Server**: Menggunakan wrapper sshagent untuk menyediakan kredensial SSH. Perintah ssh dijalankan dalam sh step untuk terhubung ke server target dan menjalankan serangkaian perintah Docker (pull, stop, rm, run) untuk memperbarui aplikasi. StrictHostKeyChecking=no ditambahkan untuk menyederhanakan koneksi pertama kali (dalam produksi, pertimbangkan cara yang lebih aman untuk menangani host key). || true ditambahkan pada docker stop dan docker rm agar perintah tidak gagal jika kontainer belum ada.
- **post { ... }**: Blok yang mendefinisikan aksi setelah pipeline selesai (selalu, sukses, gagal, dll.). cleanWs() digunakan untuk membersihkan workspace setelah build.

Langkah 5: Menjalankan Pipeline

1. Commit dan push file Jenkinsfile ke repository GitHub.
2. Kembali ke halaman job Jenkins (**nginx-cicd-pipeline**).
3. Klik Build Now di menu sebelah kiri.
4. Jenkins akan mendeteksi Jenkinsfile di repository, membaca instruksinya, dan mulai menjalankan stage satu per satu.
5. Progres pipeline dapat dipantau melalui tampilan "Stage View" atau dengan melihat "Console Output" dari build yang sedang berjalan.



Verifikasi:

- **Build & Scan:** Periksa Console Output untuk melihat log build Docker dan hasil pemindaian Trivy. Jika Trivy menemukan kerentanan HIGH/CRITICAL, pipeline akan berhenti di stage Scan.
- **Push:** Jika Scan berhasil, periksa akun Docker Hub untuk memastikan image baru dengan tag nomor build dan latest telah didorong.
- **Deploy:** Jika Push berhasil, akses server target menggunakan browser di alamat http://<IP_ADDRESS_SERVER_TARGET>:<APP_PORT> (misal: http://34.101.223.209:9090). Seharusnya halaman Nginx kustom yang dibuat di Langkah 1 ditampilkan. Periksa juga di server target dengan docker ps untuk memastikan kontainer baru berjalan.

Setelah beberapa kali troubleshooting dan penyesuaian, pipeline berhasil dijalankan.

#18

Success 2 min 7 sec ago in 22 sec

- Checkout SCM
- 1. Checkout Code
- 2. Build Docker Image
- 3. Scan Image with Trivy
- 4. Push Image to Docker Hub
- 5. Deploy to Target Server
- Post Actions

3. Scan Image with Trivy — 0.3 sec

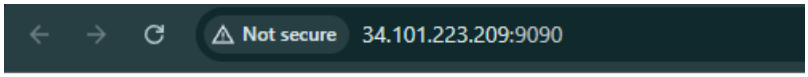
Memindai image avimajid/simple-nginx-app:19 dengan Trivy...

trivy image --exit-code 1 --severity HIGH,CRITICAL --ignore-unfixed avimajid/simple-nginx-app:19

```
+ trivy image --exit-code 1 --severity HIGH,CRITICAL --ignore-unfixed avimajid/simple-nginx-app:19
2025-04-24T16:36:52Z    INFO    [vuln] Vulnerability scanning is enabled
2025-04-24T16:36:52Z    INFO    [secret] Secret scanning is enabled
2025-04-24T16:36:52Z    INFO    [secret] If your scanning is slow, please try '--scanners vuln' to disable secret scanning
2025-04-24T16:36:52Z    INFO    [secret] Please see also https://trivy.dev/v0.61/docs/scanner/secret#recommendation for faster secret detection
2025-04-24T16:36:52Z    INFO    Detected OS     family="alpine" version="3.21.3"
2025-04-24T16:36:52Z    INFO    [alpine] Detecting vulnerabilities...   os_version="3.21" repository="3.21" pkg_num=68
2025-04-24T16:36:52Z    INFO    Number of language-specific files       num=0

Report Summary

┌──────────────────────────────────────────────┬────────┬─────────────────┬─────────┐
│                    Target                    │  Type  │ Vulnerabilities │ Secrets │
├──────────────────────────────────────────────┼────────┼─────────────────┼─────────┤
│ avimajid/simple-nginx-app:19 (alpine 3.21.3) │ alpine │        0        │    -    │
└──────────────────────────────────────────────┴────────┴─────────────────┴─────────┘
Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)
```



Halo dari Pipeline CI/CD Jenkins!

Aplikasi ini dideploy secara otomatis.