Ubuntu Makinesine Docker Kurulumu, SSH Sertifikası Oluşturma ve Jenkins Kurulumu

1. Ubuntu Makinesine Docker Kurulumu

1. Docker'ı güncel tutmak için eski sürümleri kaldırın:

   ```
   sudo apt-get remove docker docker-engine docker.io containerd runc
   ```

2. Gerekli bağımlılıkları yükleyin:

   ```
   sudo apt-get update
   sudo apt-get install \
   ca-certificates \
   curl \
   gnupg \
   lsb-release
   ```

3. Docker'ın resmi GPG anahtarını ekleyin:

   ```
   sudo mkdir -p /etc/apt/keyrings
   curl -fsSL https://download.docker.com/linux/ubuntu/gpg |
   sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
   ```

4. Docker deposunu ayarlayın:

   ```
   echo \ "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg]
   https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
   ```

5. Docker'ı yükleyin:

   ```
   sudo apt-get update
   sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
   docker-compose-plugin
   ```

6. Docker'ı çalıştırın:

   ```
   sudo systemctl start docker
   sudo systemctl enable docker
   ```

7. Docker'ı sudo olmadan çalıştırmak için kullanıcınızı docker grubuna ekleyin:

   ```
   sudo usermod -aG docker $USER
   ```

2. SSH Sertifikası Oluşturma

1. SSH anahtar çifti oluşturun:

   ```
   ssh-keygen -t rsa -b 4096 -C "admin@sampas.com.tr"
   ```

   İstendiğinde dosya ismini ve şifresini girin. Varsayılan olarak kabul edebilirsiniz. (~/.ssh/id_rsa).

2. SSH sunucusunu başlatın (Ubuntu'da):

   ```
   sudo systemctl start ssh
   sudo systemctl enable ssh
   ```

3. Genel anahtarı (public key) kopyalayın:

   ```
   cat ~/.ssh/id_rsa.pub
   ```

3. Windows Makinenize Jenkins Kurulumu

Jenkins'i Windows makinenize kurmak için aşağıdaki adımları izleyin:

3.1 Java JDK Kurulumu

Jenkins'in çalışabilmesi için Java Development Kit (JDK) yüklü olmalıdır. Adımlar aşağıdaki gibidir:

1. Oracle JDK veya OpenJDK sayfasından JDK'yı indirin ve kurun (En az Java 11 sürümü önerilir).
2. Kurulum tamamlandıktan sonra, JDK'nın yüklü olduğu dizini bulun. Genellikle bu dizin şudur: `C:\Program Files\Java\jdk-<sürüm>`
3. **Çevresel değişkenleri ayarlayın:**
   - **Java HOME'u Ayarlama:**
     1. Başlat menüsüne `Environment Variables` yazarak **Sistem Özellikleri** penceresini açın.
     2. **Gelişmiş** sekmesine gidin ve **Environment Variables...** butonuna tıklayın.
     3. **System Variables** bölümünde **New...** butonuna tıklayın.
     4. **Variable name** alanına `JAVA_HOME` yazın.
     5. **Variable value** alanına JDK'nın kurulu olduğu dizini yazın (örneğin, `C:\Program Files\Java\jdk-<sürüm>`).
   - **Path Değişkenini Ayarlama:**
     1. Aynı **Environment Variables** penceresinde **System Variables** bölümünden **Path** değişkenini seçin ve **Edit...** butonuna tıklayın.
     2. **New** butonuna tıklayın ve JDK'nın `bin` klasör yolunu ekleyin (örneğin, `C:\Program Files\Java\jdk-<sürüm>\bin`).
     3. Değişiklikleri kaydedin ve **OK** butonlarına basarak tüm pencereleri kapatın.

3.2 Jenkins İndirme ve Kurulum

1. Jenkins Windows Installer sayfasından Jenkins'in en son sürümünü indirin.
2. İndirilen `.msi` dosyasını çalıştırın ve kurulum adımlarını izleyin.
3. Kurulum sırasında varsayılan ayarları kullanabilir veya ihtiyacınıza göre özelleştirebilirsiniz.
4. Jenkins kurulduktan sonra, hizmet olarak otomatik başlatılacaktır.

3.3 Jenkins İlk Yapılandırma

1. Kurulum tamamlandıktan sonra tarayıcınızda Jenkins'e erişmek için şu adresi kullanın: `http://localhost:8080`
2. İlk giriş için kurulum sırasında verilen şifreyi kullanmanız gerekecek. Şifreyi bulmak için şu dizine gidin:

```
C:\Program Files\Jenkins\secrets\initialAdminPassword
```

Bu dosyadaki şifreyi kopyalayın ve tarayıcıya yapıştırarak Jenkins'e giriş yapın. 3. Giriş yaptıktan sonra, önerilen eklentileri yükleyin ve bir yönetici hesabı oluşturun.

3.4 Jenkins ile İlgili Diğer Ayarlar

3.4.1 Jenkins'in Hizmet Olarak Yönetimi Jenkins, Windows hizmeti olarak çalıştığından, hizmet yönetim arayüzü üzerinden kontrol edilebilir. Jenkins'i durdurmak, başlatmak veya yeniden başlatmak için aşağıdaki adımları izleyin:

1. **Hizmetler Yönetim Panelini Açın:**
   - Başlat menüsüne `services.msc` yazarak Windows Hizmetleri yönetim panelini açın.
2. **Jenkins Hizmetini Bulun:**
   - Hizmetler listesinden **Jenkins** hizmetini bulun.
3. **Jenkins Hizmetini Yönetme:**
   - Jenkins hizmetine sağ tıklayın.
   - **Başlat (Start)** seçeneği ile Jenkins'i başlatabilirsiniz.
   - **Durdur (Stop)** seçeneği ile Jenkins'i durdurabilirsiniz.

- **Yeniden Başlat (Restart)** seçeneği ile Jenkins'i yeniden başlatabilirsiniz.

3.4.2 Güncellemeler ve Yedeklemeler Jenkins'in düzgün çalışmasını sağlamak ve olası veri kayıplarını önlemek için düzenli güncellemeler ve yedeklemeler yapmak önemlidir.

1. **Jenkins Eklentilerini Güncelleme:**
   - Jenkins ana sayfasına gidin.
   - `Manage Jenkins > Manage Plugins` menüsüne tıklayın.
   - **Updates** sekmesine geçin ve güncellenmesi gereken eklentileri kontrol edin.
   - **Download now and install after restart** butonuna tıklayarak güncellemeleri indirin ve yükleyin.
2. **Yedekleme Yapma:**
   - Jenkins yapılandırmalarını ve iş verilerini düzenli olarak yedeklemek için `ThinBackup` gibi bir eklenti kullanabilirsiniz.
   - Eklentiyi kurmak için `Manage Jenkins > Manage Plugins` menüsünden **Available** sekmesini seçin ve **ThinBackup** araması yaparak yükleyin.
   - Eklentiyi kurduktan sonra, yapılandırma ve verilerinizi düzenli olarak yedeklemek için eklentinin ayarlarını yapın.
3. **Jenkins Konfigürasyon Yedeği:**
   - Jenkins'in konfigürasyon dosyaları genellikle `C:\Program Files (x86)\Jenkins\config.xml` ve `C:\Program Files (x86)\Jenkins\jobs` klasörlerinde bulunur. Bu dosyaları ve klasörleri düzenli olarak yedeklemek, sistemin geri yüklenmesini kolaylaştırır.

Bu adımlar, Jenkins'in yönetimini ve bakımını kolaylaştıracak şekilde yapılandırmanızı sağlar.

4. SSH Sertifikasını Jenkins'te Tanımlama

Jenkins'te SSH sertifikasını tanımlamak için aşağıdaki adımları izleyin:

1. Jenkins ana sayfasında `Manage Jenkins > Manage Credentials` yolunu izleyin.
2. `(global)` seçeneğini seçin ve `Add Credentials` butonuna tıklayın.
3. `Kind` seçeneğinden **SSH Username with private key** seçeneğini seçin.
4. **Username** kısmına uzak sunucuda kullanacağımız kullanıcı adını yazın.
5. **Private Key** kısmında `Enter directly` seçeneğini seçin ve oluşturduğunuz `id_rsa` dosyasının içeriğini yapıştırın.
6. Jenkins projelerinde bu SSH kimlik bilgilerini kullanmak için proje ayarlarında **Build Environment** bölümünden **Use secret text(s) or file(s)** seçeneğini işaretleyin ve eklediğiniz SSH kimlik bilgilerini seçin.

Artık Jenkins ile SSH sertifikası kullanarak uzak sunuculara erişim sağlayabilirsiniz.

5. Docker Konfigürasyonları ve GitHub Reposu Oluşturma

5.1 EtkinlikWeb İçin Dockerfile

EtkinlikWeb projesi için React uygulamanızı Docker konteynerinde çalıştırmak için aşağıdaki Dockerfile'ı kullanabilirsiniz:

```
# Temel imaj olarak Node.js kullan
FROM node:18

# Çalışma dizini oluştur
WORKDIR /app

# Paketleri yükle
COPY package.json package-lock.json ./
RUN npm install
```

```
# Proje dosyalarını kopyala
COPY . .

#Projeyi derle
RUN npm run build

# Container'ın dinleyeceği port
EXPOSE 5173

# Uygulamayı başlat
CMD ["npm", "run", "preview", "--", "--port", "5173"]
```

5.2 Sampas_Mobil_Etkinlik İçin Dockerfile

```
# .NET Core ASP.NET runtime olarak kullanıyoruz
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app

# Zaman dilimi ayarlarını Istanbul'a ayarlıyoruz
# Oracle da zaman dilimi hatası verdiği için yapılmıştır.
RUN apt-get update && apt-get install -y tzdata \
    && ln -snf /usr/share/zoneinfo/Europe/Istanbul /etc/localtime \
    && echo "Europe/Istanbul" > /etc/timezone

# Port ayarlarını yapıyoruz
EXPOSE 5262
EXPOSE 7218

# SDK imajını kullanarak uygulamayı derliyoruz
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY ["Sampas_Mobil_Etkinlik.csproj", "./"]
RUN dotnet restore "Sampas_Mobil_Etkinlik.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "Sampas_Mobil_Etkinlik.csproj" -c Release -o /app/build

# Uygulamayı publish ediyoruz
FROM build AS publish
RUN dotnet publish "Sampas_Mobil_Etkinlik.csproj" -c Release -o /app/publish

# Final aşamasında runtime imajını kullanarak çalıştırıyoruz
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .

# Ortam değişkenlerini ayarlıyoruz
ENV ASPNETCORE_URLS=http://*:5262;https://*:7218
ENV ASPNETCORE_ENVIRONMENT=Development

ENTRYPOINT ["dotnet", "Sampas_Mobil_Etkinlik.dll"]
```

5.3 GitHub Reposu Oluşturma

EtkinlikWeb ve Sampas_Mobil_Etkinlik projeleri için GitHub üzerinde repolar oluşturun

6. Jenkins Pipeline Konfigürasyonları

6.1 EtkinlikWeb Pipeline

Aşağıdaki pipeline, EtkinlikWeb projesinin Docker imajını oluşturur, etiketler ve Docker registry'ye gönderir. Ayrıca mevcut bir konteyner varsa durdurur ve kaldırır, ardından yeni imajı çekip çalıştırır.

```
pipeline {
    agent any

    stages {
        stage('Git Check') {
            steps {
                git branch: 'main', credentialsId: "github-ro-access",
                url: 'git@github.com:Ebu13/EtkinlikWeb.git'
            }
        }
        stage('Image Build & Push') {
            steps {
                sh 'docker build -t etkinlik-web:latest .'
                sh 'docker tag etkinlik-web:latest
                registry.sampas.com.tr/commons/onlinebelediye/etkinlik-web:latest'
                sh 'docker push
                registry.sampas.com.tr/commons/onlinebelediye/etkinlik-web:latest'
            }
        }
        stage('Clean Up') {
            steps {
                sh 'ssh -t -t root@192.168.34.13 "docker stop etkinlik-web || true"'
                sh 'ssh -t -t root@192.168.34.13 "docker rm etkinlik-web || true"'
            }
        }
        stage('Release') {
            steps {
                sh 'ssh -t -t root@192.168.34.13 "docker pull
                registry.sampas.com.tr/commons/onlinebelediye/etkinlik-web:latest"'
                sh 'ssh -t -t root@192.168.34.13 "docker run --name=etkinlik-web
                -d --restart=always -p 5173:5173
                registry.sampas.com.tr/commons/onlinebelediye/etkinlik-web:latest"'
            }
        }
    }
}
```

İsterseniz kendi projeniz için:

```
pipeline {
    agent any

    stages {
        stage('Git Check') {
            steps {
```

```
                git branch: '{branch ismi}', credentialsId: "github-ro-access",
                url: '<GIT_REPOSITORY_URL>'
            }
        }
        stage('Image Build & Push') {
            steps {
                sh 'docker build -t <DOCKER_IMAGE_NAME>:latest .'
                sh 'docker tag <DOCKER_IMAGE_NAME>:latest
                <DOCKER_REGISTRY_URL>/<DOCKER_IMAGE_NAME>:latest'
                sh 'docker push <DOCKER_REGISTRY_URL>/<DOCKER_IMAGE_NAME>:latest'
            }
        }
        stage('Clean Up') {
            steps {
                sh 'ssh -t -t <REMOTE_USER>@<REMOTE_HOST>
                "docker stop <CONTAINER_NAME> || true"
                '
                sh 'ssh -t -t <REMOTE_USER>@<REMOTE_HOST>
                "docker rm <CONTAINER_NAME> || true"
                '
            }
        }
        stage('Release') {
            steps {
                sh 'ssh -t -t <REMOTE_USER>@<REMOTE_HOST>
                "docker pull <DOCKER_REGISTRY_URL>/<DOCKER_IMAGE_NAME>:latest"
                '
                sh 'ssh -t -t <REMOTE_USER>@<REMOTE_HOST>
                "docker run --name=<CONTAINER_NAME> -d --restart=always <PORT_MAPPINGS>
                <DOCKER_REGISTRY_URL>/<DOCKER_IMAGE_NAME>:latest"
                '
            }
        }
    }
}
```

6.2 Sampas_Mobil_Etkinlik Pipeline

Aşağıdaki pipeline, Sampas_Mobil_Etkinlik projesinin Docker imajını oluşturur, etiketler ve Docker registry'ye gönderir. Ayrıca mevcut bir konteyner varsa durdurur ve kaldırır, ardından yeni imajı çekip çalıştırır.

```
pipeline {
    agent any

    stages {
        stage('Git Check') {
            steps {
                git branch: 'main', credentialsId: "github-ro-access",
                url: 'git@github.com:Ebu13/Sampas_Mobil_Etkinlik.git'
            }
        }
        stage('Image Build & Push') {
            steps {
                sh 'docker build -t sampas-mobil-etkinlik:latest .'
                sh 'docker tag sampas-mobil-etkinlik:latest
```

```
                registry.sampas.com.tr/commons/onlinebelediye/sampas-mobil-etkinlik:latest'
                sh 'docker push
                registry.sampas.com.tr/commons/onlinebelediye/sampas-mobil-etkinlik:latest'
            }
        }
        stage('Clean Up') {
            steps {
                sh 'ssh -t -t root@192.168.34.13 "docker stop sampas-mobil-etkinlik || true"'
                sh 'ssh -t -t root@192.168.34.13 "docker rm sampas-mobil-etkinlik || true"'
            }
        }
        stage('Release') {
            steps {
                sh 'ssh -t -t root@192.168.34.13
                "docker pull
                registry.sampas.com.tr/commons/onlinebelediye/sampas-mobil-etkinlik:latest"
                '
                sh 'ssh -t -t root@192.168.34.13
                "docker run --name=sampas-mobil-etkinlik -d --restart=always -p
                5262:5262 -p 7218:7218
                registry.sampas.com.tr/commons/onlinebelediye/sampas-mobil-etkinlik:latest"'
            }
}
}
}
```

isterseniz kendi projeniz için:

```
pipeline {
    agent any
    /*
    parameters {
        string(name: 'GIT_BRANCH', defaultValue: 'main',
        description: 'Git branch to build from.')
        string(name: 'GIT_CREDENTIALS_ID', defaultValue: 'github-ro-access',
        description: 'ID of the Git credentials to use.')
        string(name: 'GIT_URL', defaultValue: 'git@github.com:Ebu13/Sampas_Mobil_Etkinlik.git',
        description: 'URL of the Git repository.')
        string(name: 'IMAGE_NAME', defaultValue: 'sampas-mobil-etkinlik',
        description: 'Name of the Docker image.')
        string(name: 'IMAGE_TAG', defaultValue: 'latest',
        description: 'Tag for the Docker image.')
        string(name: 'REGISTRY_URL', defaultValue: 'registry.sampas.com.tr',
        description: 'URL of the Docker registry.')
        string(name: 'REGISTRY_REPO', defaultValue: 'commons/onlinebelediye',
        description: 'Docker registry repository path.')
        string(name: 'DEPLOY_HOST', defaultValue: '192.168.34.13',
        description: 'Host for deployment.')
        string(name: 'DEPLOY_PORT', defaultValue: '22',
        description: 'Port for SSH connection.')
        string(name: 'DEPLOY_NAME', defaultValue: 'sampas-mobil-etkinlik',
        description: 'Name of the Docker container.')
        string(name: 'DEPLOY_PORTS', defaultValue: '5262:5262,7218:7218',
        description: 'Ports to expose on the Docker container.')
```

```
    }
    */
    stages {
        stage('Git Check') {
            steps {
                git branch: "${params.GIT_BRANCH}",
                credentialsId: "${params.GIT_CREDENTIALS_ID}", url: "${params.GIT_URL}"
            }
        }
        stage('Image Build & Push') {
            steps {
                sh "docker build -t ${params.IMAGE_NAME}:${params.IMAGE_TAG} ."
                sh "docker tag ${params.IMAGE_NAME}:${params.IMAGE_TAG}
                ${params.REGISTRY_URL}/
                ${params.REGISTRY_REPO}/
                ${params.IMAGE_NAME}:
                ${params.IMAGE_TAG}"
                sh "docker push ${params.REGISTRY_URL}/
                ${params.REGISTRY_REPO}/
                ${params.IMAGE_NAME}:${params.IMAGE_TAG}"
            }
        }
        stage('Clean Up') {
            steps {
                sh "ssh -p ${params.DEPLOY_PORT} root@${params.DEPLOY_HOST}
                'docker stop ${params.DEPLOY_NAME} || true'"
                sh "ssh -p ${params.DEPLOY_PORT} root@${params.DEPLOY_HOST}
                'docker rm ${params.DEPLOY_NAME} || true'"
            }
        }
        stage('Release') {
            steps {
                sh "ssh -p ${params.DEPLOY_PORT} root@${params.DEPLOY_HOST}
                'docker pull ${params.REGISTRY_URL}/${params.REGISTRY_REPO}/
                ${params.IMAGE_NAME}:${params.IMAGE_TAG}'"
                sh "ssh -p ${params.DEPLOY_PORT} root@${params.DEPLOY_HOST}
                'docker run --name=${params.DEPLOY_NAME} -d --restart=always -p
                ${params.DEPLOY_PORTS} ${params.REGISTRY_URL}/
                ${params.REGISTRY_REPO}/
                ${params.IMAGE_NAME}:${params.IMAGE_TAG}'"
            }
        }
    }
}
```

Bu iki pipeline ı da oluşturduktan sonra build ettiğimizde proje çalışacaktır.

6.3 DevOps Süreci ve Kullanımı

Bu DevOps pipeline'ı, Sampas_Mobil_Etkinlik projenizde Docker imajını otomatik olarak oluşturur, registry'ye yükler ve yeni imajı kullanarak bir konteyner çalıştırır. Bu işlemler sayesinde, geliştirme sürecinde sürekli entegrasyon ve teslimat (CI/CD) sağlanmış olur. DevOps sürecinizi verimli bir şekilde nasıl kullanabileceğinizi aşağıda detaylandırdım:

1. Geliştirme Süreci Kod Güncellemeleri: Projenizde bir değişiklik yapmak istediğinizde, bu değişiklikleri yerel olarak geliştirip test ettikten sonra GitHub'daki Sampas_Mobil_Etkinlik repository'sine commit ve push yapabilirsiniz.

GitHub Commit ve Push:

Örneğin, bir bug fix veya yeni özellik geliştirdiniz. Yerel projenizde gerekli değişiklikleri yaptıktan sonra, şu adımları izleyebilirsiniz:

```
git add .
git commit -m "Yeni özellik eklendi"
git push origin main
```

2. Jenkins Pipeline Tetikleme Otomatik Tetikleme: Jenkins, GitHub repository'sindeki değişiklikleri izleyebilir ve main branch'e her yeni commit yapıldığında pipeline'ı otomatik olarak tetikleyebilir. Bu sayede manuel bir işlem yapmadan her kod değişikliğinde projeniz için yeni bir Docker imajı oluşturulur ve yayınlanır. Pipeline Süreci: Pipeline, şu aşamaları içerir: Git Check: Projenizin en güncel halini GitHub'dan çeker. Image Build & Push: Docker imajını oluşturur, registry'ye yükler. Clean Up: Mevcut çalışan konteyneri durdurur ve kaldırır. Release-Dev: Yeni imajı çeker ve konteyneri yeniden başlatır.

3. Canlı Ortama Entegrasyon Test ve QA Süreci: Jenkins pipeline'ınızın başarılı olması durumunda, imajınız geliştirme ortamında otomatik olarak devreye alınır. Geliştirme ve test ortamlarında yaptığınız testlerden sonra, gerekirse bu imajı canlı ortama taşıyabilirsiniz.

Canlı Ortama Yayınlama: Geliştirme ortamında başarılı bir şekilde test edilen imajı canlı ortama yayınlayabilirsiniz. Bu adım, Jenkins üzerinden bir production pipeline veya manuel bir deployment süreci ile yönetilebilir.

4. Sürekli Entegrasyon (CI) Sürekli Entegrasyon: Her kod değişikliğinde otomatik olarak tetiklenen Jenkins pipeline'ı sayesinde, kodunuz sürekli olarak entegre edilir ve imajlarınız güncel kalır. Bu süreç, ekip içinde işbirliğini artırır ve kod kalitesini yükseltir.
5. Pipeline Yönetimi Pipeline İzleme: Jenkins dashboard'undan pipeline'ın her aşamasını izleyebilirsiniz. Olası hataları görmek ve pipeline'ı optimize etmek için loglara erişim sağlayabilirsiniz.

Manuel Tetikleme: İhtiyaç duyduğunuzda pipeline'ı manuel olarak da tetikleyebilir ve belirli aşamaları yeniden çalıştırabilirsiniz.

6. Hata Yönetimi Olası Hatalar: Eğer pipeline bir aşamada durur veya hata verirse, Jenkins'in sunduğu log ve hata mesajlarını inceleyerek hızlıca müdahale edebilirsiniz. Örneğin, Docker imajı oluşturulamazsa veya konteyner başlatılamazsa, ilgili komutların loglarını kontrol ederek sorunu çözebilirsiniz. Sonuç Bu süreç, geliştirme hızınızı artırır, hataları minimuma indirir ve ekip içinde daha iyi bir işbirliği sağlar. DevOps araçlarını ve bu pipeline'ı kullanarak Sampas_Mobil_Etkinlik projenizi sürekli entegrasyon ve teslimat (CI/CD) prensiplerine uygun şekilde yönetebilirsiniz. Bu sayede, her kod değişikliğinde proje otomatik olarak test edilir, inşa edilir ve devreye alınır.