

Okuma: Docker Kavramlarının Gözden Geçirilmesi ve Dockerfile'ı Anlama

Tahmini süre: 30 dakika

Bu okuma, Docker kavramlarını anlamınızı geliştirmek ve bir Dockerfile'ı anlama sürecinde size rehberlik etmek için tasarlanmıştır.

Okuma Genel Görünümü:

- Docker Kavramları
- Dockerfile Anlamak

Docker Kavramları

Genel Bakış

Docker, uygulamaların oluşturulmasını, dağıtılmmasını ve yönetilmesini konteynerler kullanarak basitleştirir. Konteynerler, bir uygulamayı bağımlılıklarıyla birlikte yazılım geliştirme için standartlaşdırılmış bir birim olarak paketlemenizi sağlar. Docker, çeşitli ortamlarda konteynerleri oluşturmak, göndermek ve çalıştmak için araçlar ve bir platform sunar.

Dockerfile

Docker görüntüsü oluşturmak için talimatlar içeren bir metin dosyasıdır. Temel görüntüyü belirtir, çalışma dizinini ayarlar, bağımlılıkları yükler, uygulama kodunu kopyalar, portları açar ve uygulamayı çalıştmak için komutlar tanımlar.

Konteyner

Bir Docker görüntüsünün, ana makinede bir işlem olarak çalışan bir örneğidir. Konteynerler hafif, taşınabilir ve izole olduğundan, uygulamaları dağıtmak ve ölçeklendirmek için idealdir.

Docker Görüntü Depolama

Docker görüntüleri, kamuya açık veya özel olabilen kayıt defterlerinde saklanır. Docker Hub gibi kamuya açık kayıt defterleri milyonlarca görüntü barındırırken, kuruluşlar genellikle güvenlik ve kontrol amacıyla özel kayıt defterlerini kullanır.

Docker Görsellerinin Nerede Bulunduğu

Yerel Makine: Bir Docker görseli oluşturduğunuzda, başlangıçta yerel olarak makinenizde depolanır. Yerel Docker görsellerini listelemek için docker images komutunu kullanabilirsiniz.

Kayıt Defteri: Oluşturduktan sonra, Docker görsellerini bir kayıt defterine yükleyebilir, böylece bu kayıt defterine erişimi olan her yerden erişilebilir hale getirebilirsiniz.

Dockerfile

Aşağıda referans için bir örnek Dockerfile bulunmaktadır. Ardından gelen açıklama, kullanılan komutları detaylandıracak bir Dockerfile yazma konusunda kapsamlı bir anlayış sunmaktadır.

▼ Dockerfile

```
# Use the official Node.js image as the base image
FROM node:14
# Set environment variables
ENV NODE_ENV=production
ENV PORT=3000
# Set the working directory
WORKDIR /app
# Copy package.json and package-lock.json files
COPY package*.json .
# Install dependencies
RUN npm install --production
# Copy the rest of the application code
COPY .
# Add additional file
ADD public/index.html /app/public/index.html
# Expose the port on which the application will run
EXPOSE $PORT
# Specify the default command to run when the container starts
CMD ["node", "app.js"]
# Labeling the image
LABEL version="1.0"
LABEL description="Node.js application Docker image"
LABEL maintainer="Your Name"
# Healthcheck to ensure the container is running correctly
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 CMD curl -fs http://localhost:$PORT || exit 1
# Set a non-root user for security purposes
USER node
```

Dockerfile'ı Anlamak

1: Temel Görüntüyü Belirleyin

Başlamak için, `FROM` talimatını kullanarak temel görüntüyü belirtin. Bu durumda, resmi Node.js görüntü sürümü 14 kullanılmaktadır.

```
FROM node:14
```

`FROM`: Docker konteyneri için temel görüntüyü belirtir.

`node:14`: Docker kayıt defterinden sürüm 14 ile resmi Node.js görüntüsünü çeker.

2: Ortam Değişkenlerini Ayarlayın

Sonra, gerekli ortam değişkenlerini `ENV` talimatı ile tanımlayın. Burada, `NODE_ENV` üretim olarak ayarlanmış ve `PORT` 3000 olarak ayarlanmıştır.

```
ENV NODE_ENV=production  
ENV PORT=3000
```

`ENV`: Docker konteyneri içinde ortam değişkenlerini ayarlar.

`NODE_ENV=production`: Node.js ortamını üretim moduna ayarlar.

`PORT=3000`: Node.js uygulamasının dinleyeceği portu ayarlar.

3: Çalışma Dizini Ayarlama

Ardından, `WORKDIR` talimatını kullanarak konteyner içindeki çalışma dizinini belirtin. Bu, `/app`'i çalışma dizini olarak ayarlar.

```
WORKDIR /usr/src/app
```

`WORKDIR`: Ardışık talimatlar için çalışma dizinini tanımlar, dosya yolu referanslarını basitleştirir.

`/usr/src/app`: Uygulama kodunuzu depolamak için konteyner içindeki dizin, düzenli ve kolay erişilebilir tutar.

4: Paket Dosyalarını Kopyala

Çalışma dizinini tanımladıktan sonra, uygulama bağımlılıklarını yüklemek için gerekli olan `package.json` ve `package-lock.json` dosyalarını konteynır kopyalamak için `COPY` talimatını kullanın.

```
COPY package*.json ./
```

`COPY`: Dosyaları yerel makinenizden konteynır kopyalar.

`package*.json ./`: Hem `package.json` hem de `package-lock.json` dosyalarını kopyalar.

5: Bağımlılıkları Yükleyin

Paket dosyaları kopyalandıktan sonra, `package.json` dosyasında listelenen üretim bağımlılıklarını yüklemek için `RUN` talimatını çalıştırın.

```
RUN npm install --production
```

- **ÇALIŞTIR:** Yapı sürecinde konteyner içinde komutları yürütür.
- **npm install –production:** package.json dosyasında belirtilen bağımlılıkları devDependencies olmadan yükler.

6: Uygulama Kodunu Kopyala

Bağımlılıklar yüklenikten sonra, uygulamanın geri kalan kodunu COPY talimatını kullanarak konteynırı kopyalayın.

```
COPY . .
```

COPY . .: Mevcut dizindeki tüm dosyaları ve dizinleri yerel ana bilgisayardan Docker konteynerindeki mevcut dizine kopyalar.

7: Ek Dosya(lar) Ekle

Ayrıca, ekstra dosyaları eklemek için ADD talimatını kullanın. Burada, index.html dosyası kamu dizinine eklenir.

```
# ADD <source_path> <destination_path>
ADD public/index.html /app/public/index.html
```

- **public/index.html:** Ana makinedeki dosya veya dizinin yolu.
- **/app/public/index.html:** Docker imajı içinde dosya veya dizini eklemek istediğiniz yol.

8: Uygulama Portunu Açıma

Sonra, Docker'a konteynerin belirtilen portta dinlediğini EXPOSE talimatı ile bildirin. Burada, \$PORT ortam değişkeni tarafından tanımlanan port açılır.

```
EXPOSE $PORT
```

- **EXPOSE:** Docker'a konteynerin çalışma zamanında belirtilen portta dinleyeceğini bildirir.
- **\$PORT:** Daha önce tanımlanan port numarasını temsil eden ortam değişkenidir.

9: Varsayılan Komutu Belirle

Konteyner başladığında çalışacak varsayılan komutu CMD talimatını kullanarak tanımlayın. Burada, node app.js çalıştırılır.

```
CMD ["node", "app.js"]
```

CMD: Bu talimat, konteynerin giriş noktasında yürütülecek varsayılan komutu ve/veya parametreleri belirtir.
["node", "app.js"]: Bu dizi, herhangi bir argüman için yürütülecek komutu belirtir. Bu durumda, Docker'a app.js argümanı ile node komutunu yürütmesini söyler.

Not: app.js dosya adı, Node.js uygulamaları için genellikle ana dosya olarak kullanılır. Ancak, proje yapınızı veya adlandırma kurallarınıza bağlı olarak, ana dosyanın farklı bir adı olabilir.

10: Görüsel Etiketle

Ardından, LABEL talimatını kullanarak görsele sürüm, açıklama ve bakım bilgilerini içeren meta veriler ekleyin.

```
LABEL version="1.0"
LABEL description="Node.js application Docker image"
LABEL maintainer="Your Name"
```

LABEL: Docker görüntüsüne meta veri ekler.
version="1.0": Docker görüntüsünün sürümünü belirtir.
description="Node.js application Docker image": Docker görüntüsünün tanımını sağlar.
maintainer="Your Name": Docker görüntüsünün bakımını üstlenen kişiyi belirtir.

11: Bir Sağlık Kontrolü Ekleyn

Konteynerin doğru çalıştığından emin olmak için, **HEALTHCHECK** talimatını kullanarak bir sağlık kontrolü tanımlayın. Bu komut, belirtilen porta bir istek göndererek uygulamanın sağlığını kontrol eder.

```
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 CMD curl -fs http://localhost:$PORT || exit 1
```

HEALTHCHECK: Konteynerin doğru çalıştığından emin olmak için bir sağlık kontrolü yapılandırır.

--interval=30s: Sağlık kontrolleri arasındaki aralığı belirtir.
--timeout=10s: Her sağlık kontrolü için zaman aşısını ayarlar.
--start-period=5s: Sağlık kontrollerinin başlamasından önce konteynerin başlatılması gereken başlangıç süresini tanımlar.
--retries=3: Konteynerin sağıksız olarak kabul edilmeden önceki deneme sayısını ayarlar.
CMD curl -fs http://localhost:\$PORT || exit 1: Sağlık kontrolleri için çalıştırılacak komutu belirtir. http://localhost:\$PORT adresine yapılan bir isteğin başarılı olup olmadığını kontrol eder; aksi takdirde, başarısızlık durumunu belirten 1 koduyla çıkar.

12: Kök Olmayan Bir Kullanıcı Ayarlayın

Son olarak, güvenlik amacıyla, **USER** talimatını kullanarak kök olmayan bir kullanıcı ayarlayın. Burada, kullanıcı node olarak ayarlanmıştır.

```
USER node
```

KULLANICI: Dockerfile'daki sonraki talimatları çalıştıracak kullanıcıyı ayarlar.

node: Güvenlik amacıyla komutları çalıştırmak için node adlı kullanıcıyı belirtir.

Bu adım adım açıklama, bir Dockerfile oluşturma sürecini gösterir ve her bir talimatın, tam işlevsel bir Docker imajı oluşturmak için nasıl önceki talimatın üzerine inşa edildiğini vurgular.

Özet

Bu okumada, Docker kavramları ve bir Dockerfile oluşturma süreci hakkında bilgi edindiniz. Sağlanan Dockerfile ve özetlenen adımlar, bir Node.js uygulaması için bir Docker imajı oluşturanın nasıl yapılacağını göstermektedir. Resmi Node.js temel imajı ile başlayarak, ortam değişkenleri ayarlanır, bir çalışma dizini tanımlanır ve gerekli bağımlılıklar yüklenir. Uygulama kodu daha sonra konteynırına kopyalanır ve konteynır düzgün çalıştığından emin olmak için bir sağlık kontrolü eklenir. Son olarak, güvenliği artırmak için bir kök olmayan kullanıcı ayarlanır.

Yazar: [Nikesh Kumar](#)



Skills Network