Konteyner Tabanlı Sanallaştırma ve 12 Faktör Yöntembilimine Dayalı Web Uygulama Geliştirme Süreci Önerisi

Emre Can YILMAZ[1], Recai OKTAŞ[2]

[1]Ondokuz Mayıs Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı, Samsun

[2]Ondokuz Mayıs Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Samsun

ecylmz@bil.omu.edu.tr, roktas@bil.omu.edu.tr

Özet

- Günümüzde, konteyner tabanlı sanallaştırmanın popüler hale gelmesiyle birlikte uygulama geliştirme yöntemlerinde de değişikliğe gidilmektedir.

- Geliştiriciler ve sistem yöneticileri tarafından, geliştirme ve konuşlandırma süreçleri farklılıklarını minimuma indirmek istenmektedir.

- Bunun yanı sıra geliştiricilerin çalıştıkları ortam farklılıkları da ortadan kaldırmak istenmektedir.

Özet

- Bu çalışmada, geliştirme sürecindeki kritik olan bu problemlere modern çözüm önerisi getirilmiştir.

- Konteyner tabanlı sanallaştırma olarak Docker ve örnek web uygulaması olarak Ruby on Rails web çatısı kullanılmıştır.

Giriş

- Modern çağda, yazılımlar genel olarak servis bazında hizmet vermektedir. Bu yazılımlar; web uygulamaları ya da SaaS(software-as-a-service) olarak anılmaktadır.

- 12 Faktör uygulaması web uygulamaları inşa edebilmek için kullanılan, toplamda 12 ana unsurdan oluşan bir yöntembilimdir. Bu yöntem ile birlikte şu sorunlara çözüm getirmektedir:

12 Faktör Faydaları

- Yazılımın kurulumunun otomatize edilmesi ile birlikte, kurulum esnasında harcanacak vakit en aza indirgenmekle birlikte, yeni bir geliştiricisinin projeye dahil olma maliyeti en aza indirgenir.

- Farklı uygulama çalıştırma ortamları arasındaki geçişlerde sorunu en aza indirger.

- Modern bulut bilişim platformlarında konuşlandırma için uygun olup, sunucu ve sistem yönetimi ihtiyacını minimuma indirir.

12 Faktör Faydaları

- Geliştirme ve inşa etme aşamaları arasındaki farklılıkları minimum seviyeye getirerek, konuşlandırma için de kolaylık sağlar.

- Yazılan uygulama kullanılan araçlardan, takımlardan, mimarilerden ve geliştirme ortamlarından bağımsız olarak ölçeklenebilir.

Kullanılan Teknolojiler

- 12 faktörü en iyi şekilde uygulamak için konteyner tabanlı sanallaştırma olan Docker biçilmiş kaftan niteliği görmektedir. Docker-Compose ise Docker konteynerleri yönetiminde kullanılan araçtır.

- Bu çalışmada da 12 faktörü Ruby on Rails web çatısı ile Docker üzerinde uygulamanın niteliklerine, ayrıntılarına ve faydalarına dair inceleme bulunmaktadır.

12 Faktör Ana Unsurları

- Kod Tabanı
- Bağımlılıklar
- Yapılandırma
- Arka Plan Servisleri
- İnşa Et, Duyur, Çalıştır
- Süreçler
- Port Bağlama
- Eşzamanlılık
- Tek Kullanımlık
- Geliştirme/Sunum Benzerliği
- Günlükler
- Yönetici İşlemleri

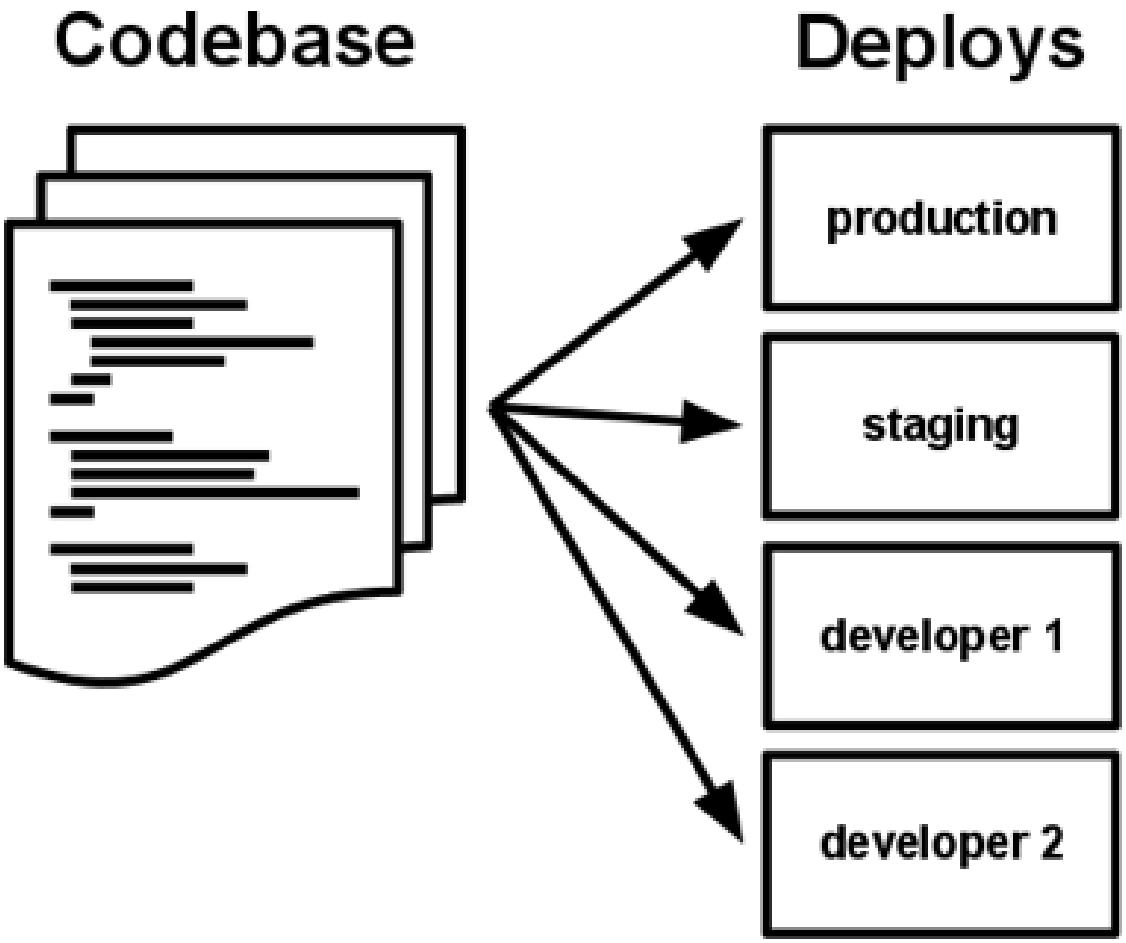
Kod Tabanı

- 12 faktör uygulamaları her zaman Git, Mercurial, Subversion gibi sürüm takip sistemleri ile izlenir.

- Aynı kod tabanının birden fazla uygulamada kullanılması 12 faktör ihlalidir.

- Her uygulamanın sadece bir tane kod tabanı olabilir ancak her uygulama birden fazla kez konuşlandırılabilir.

Kod Tabanı


Codebase
Deploys
production
staging
developer 1
developer 2

Kod Tabanı

- Docker’da ise bu maddeyi tamamlayacak olan sürüm yapısı bulunmaktadır. Docker ile konuşlandırılan her uygulamaya bir sürüm atanabilir. Sürümde hata olması durumda bir önceki sürüme dönüş yapılabilir.

- Örnek bir sürümleme için komut satırında aşağıdakine benzer komut verilmelidir.

```
$ docker build -t ecylmz/node:2.0.0 .
```

Bağımlılıklar

- Pek çok programlama dili dağıtık destek kütüphaneleri için paketleme sistemi sunmaktadır. Bu çalışmada kullanılan Ruby on Rails için bu paketleme sistemi Bundle'dır.

- Uygulamanın bağımlı olduğu tüm bağımlılıklar doğru ve tam olarak bir manifesto ile deklare edilmelidir.

- 12 Faktör uygulamaları asla sistem genelinde kullanılan paketlere güvenerek onları kullanmamalıdır.(ImageMagick, Curl)

Bağımlılıklar

- Docker'ın manifestosu olan Dockerfile adı verilen dosyada belirli formatta işlenir. Bu dosyanın her bir aşamasında gerekli bağımlılıklar, paketler yüklenir.

- Örnek bir Dockerfile satırı:

  RUN bundle install

- Yukarıdaki satırla uygulamanın istediği bağımlılıklar *bundle* komutu aracılığıyla oluşturulacak olan imaja yüklenir.

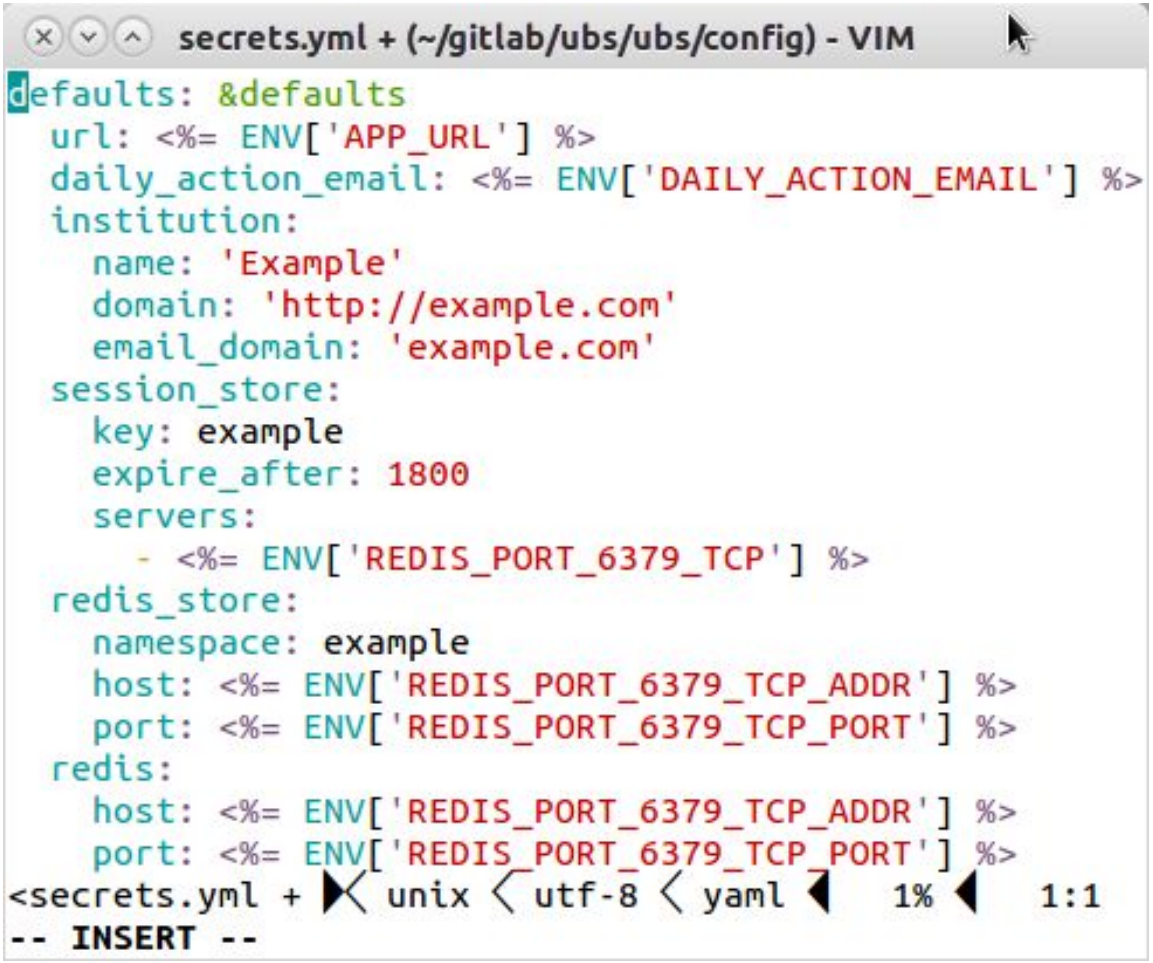
Yapılandırma

- Uygulamalar bazen yapılandırmaları kod içerisinde saklamaktadır. Bu doğru bir yaklaşım değildir ve 12 faktör kurallarına aykırıdır.

- 12 Faktör uygulamaları, yapılandırmaları çevre değişkenlerinde saklar (çoğu zaman çevre değişkenleri “env” şeklinde anılmaktadır).

- Bu çevre değişkenleri uygulama geliştirme sürecinde “docker-compose.yml” dosya içerisinde çevre değişkeni olarak ataması yapılır.

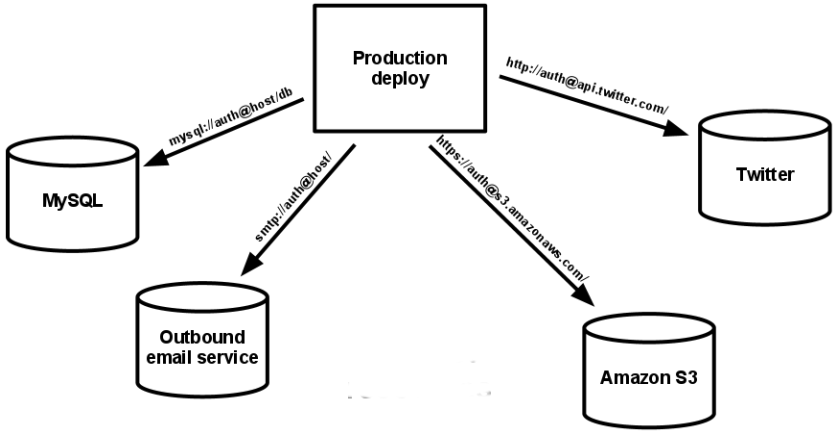
Yapılandırma



Yapılandırma - docker-compose.yml

Arka Plan Servisleri

- Örnek olarak: Veritabanları(MySQL), iş kuyruğu sistemleri(Sidekiq), SMTP sistemleri, Önbellek sistemleri
- 12 faktör uygulamaları yerel servisler ile üçüntü taraf servisler arasında ayrım yapmamaktadır. Her ikisinde de ulaşım URL veya servis kimlik bilgilerini yapılandırma ile olmaktadır.
- Docker-compose aracılığıyla arka plan servisleri sisteme kolayca entegre edilebilmektedir.



İnşa Et, Duyur, Çalıştır

İnşa etme aşaması: Bu aşamada kod tabanı inşa edilir. İnşa edilirken tüm bağımlılıklar ve gerekli olan yazılımlar sisteme dahil edilerek inşa edilmesi gerçekleştirilir.

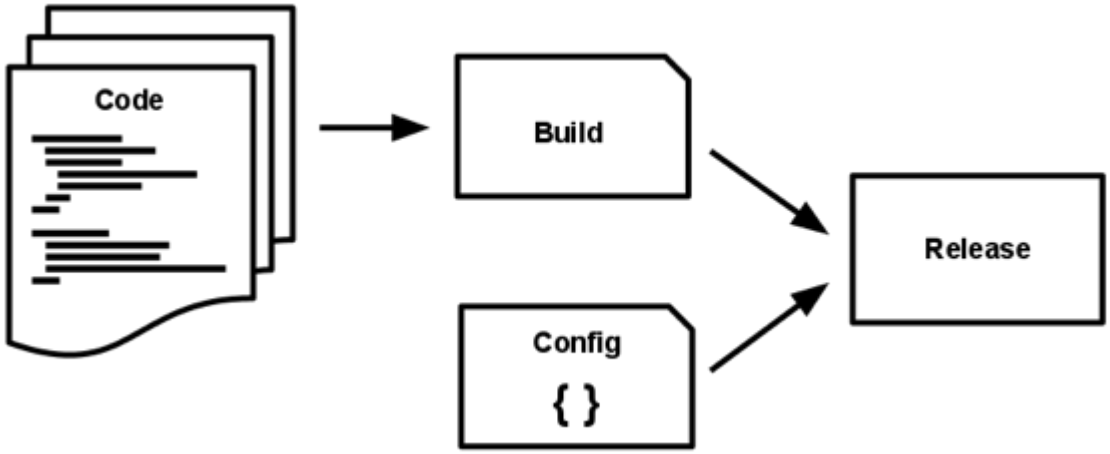
Duyuru aşaması: inşa etme sürecinin ardından uygulama içerisinde kullanılacak olan yapılandırmalar yapılır.

Çalıştırma aşaması: Diğer aşamaların ardından uygulama çalıştırma ortamında istenilen duyuru sürümünde çalıştırılarak süreçler oluşturulur.

İnşa Et, Duyur, Çalıştır

- Docker kullanarak bu 12 faktör ilkesi şu şekilde karşılanır:

- Proje içerisinde bulunan birden fazla Dockerfile bulunur. Bunlardan ilki “Dockerfile.dev” isimli olurken diğeri “Dockerfile.pro” olur. İlki geliştirme aşaması için gerekli olan imajı elde etmek için kullanılırken diğeri sunum aşamasında konuşlandırma için kullanılır. Sonuç olarak birden fazla imaj üretilir.



Süreçler

- Uygulama çalışma ortamında bir veya birden fazla süreç çalıştırabilir. 12 faktör süreçleri durumsuzdur ve paylaşımsızdır.

- Tüm veriler arka plan servislerinde kaydedilmek zorundadır bu geleneksel olarak veri tabanlarıdır.(Ruby on Rails - Redis)

- Docker-Compose aracı sayesinde bu servisler kolayca eklenebilmektedir. Örnek bir Redis servisi eklemek için docker-compose.yml dosyası içerisinde aşağıdaki kısım eklenir. Eklenen bu Redis web uygulamasına bağlanır.

  redis:

  image: redis:latest

Port Bağlama

- Bir port HTTP servisine tanımlanır ve bu uygulamada o porta gelen istekleri dinler.

- Web uygulaması için üretilen Docker konteynerine erişmek için *Dockerfile*'da web alanına şuna benzer satır eklenir:

  ports:

  - "3000:3000"

Eşzamanlılık

- Örnek verecek olunursa; HTTP istekleri ayrı bir “**web**” sürecinde çalıştırılmak istenebilir veya arka planda yapılması gerekli olan ve uzun zaman alan süreçleri “**worker**” sürecine atanmak istenebilir.

Tek Kullanımlık

- 12 faktör uygulamalarının süreçleri tek kullanımlıktır ve bu süreçler her an kolayca sonlandırılabilir veya başlatılabilir. Bu, hızlı elastik ölçekleme, kod veya yapılandırma değişikliklerini, hızlı dağıtım ve sunum konuşlandırma sağlamlığını kolaylaştırır.

- docker-compose aracılığıyla bu 12 faktör ilkesi şöyle karşılanmaktadır:

  $ docker-compose run web bundle exec rake db:build

- Yukarıdaki komut aracılığıyla mevcut imajdan yeni bir konteyner üretilip komu çalışıtırılır ve çıkılır. Bir daha bu konteyner kullanılmaz.

Geliştirme/Sunum Benzerliği

- 12 faktör uygulamaları geliştirme ve sunum aşamalarındaki farkı en aza indirgeyecek şekilde ve sürekli geliştirilecek biçimde tasarlanmıştır.

Zaman farkı: geliştirici kodu yazdıktan sonra saatler sonra değil sadece bir kaç dakika içinde konuşlandırılabilmektedir.

Personel farkı: Geliştiriciler konuşlandırma ile yakından ilgilenerek, uygulamaların sunum ortamındaki davranışlarını gözlemleyebilirler.

Araçların farkı: Geliştirme ve sunum aşamalarında benzer araçlar kullanılır.

Yukarıda bahsedilen farklılıkları gidermek için Ruby on Rails Gemfile manifestosunu sunarken, Docker Dockerfile manifestosu sunmaktadır.

Günlükler

- 12 faktör uygulamaları asla kendi çıktılarının yönlendirilmesiyle veya depolanmasıyla ilgilenmez.

- Mevcut geliştirme ortamında geliştiriciler uygulamanın çıktılarını çalıştırdıkları terminal üzerinden göreceklerdir. “***docker-compose up”*** komutuyla birlikte terminalde çıktılar görülebilir.

- Sunucu ortamında çalışılan uygulamalar, arka planda çalıştırdıkları her süreçten çıkacak olan çıktıları bir yönlendirici ile bir yerde saklamalıdır. (Logplex, Fluent).

Yönetici İşlemleri

- Geliştiricilerin yönetim amaçlı sadece bir defa verdiği komutlar bulunmaktadır. Örnek olarak veritabanı göç işlemleri veya hatalı kayıt düzeltme işlemleri olabilir.

- Tek seferlik çalıştırılacak yönetici komutları, uygulama ile aynı çevre ortamında çalıştırılmalıdır.

- Örneğin; veritabanı göç işlemleri için:

    $ docker-compose run web bundle exec rake db:migrate

Sonuç ve Öneriler

- Bu çalışma, mevcut teknolojilerinin, web uygulama geliştirme aşamalarında en iyi ve en doğru şekilde kullanımına odaklanmıştır.

- Her ne kadar bu çalışma uygulama geliştirme süreçlerine odaklansa da uygulanan adımlar uygulamanın sunucu ortamında konuşlandırılmasını da kolaylaştırdığı açıktır.

- Sunucu ortamının Docker aracılığıyla yerelde hazırlanıyor olması, olası sorunları önceden tespit etmeyi sağlamaktadır.

Sonuç ve Öneriler

- Docker ve 12 faktörün uygulanmasıyla hazırlanan uygulamaların sunucu ortamında ölçeklenmesi daha kolay olmaktadır.(docker-compose, dokku, heroku)

- Geliştiriciler artık SSH hesabı istemeyeceklerdir. Tek kullanımlık komutlar ile işlerini halledebilirler.

Kaynak Kodlar

- Bu çalışmada, şekillerde gösterilen kodlar: https://github.com/ecylmz/learn-rails-docker adresinde bulunmaktadır.

Teşekkürler!

Emre Can YILMAZ

Ondokuz Mayıs Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı, Samsun

ecylmz@bil.omu.edu.tr