

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

Emre Can Yılmaz¹, Recai Oktas²

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 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. 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.

Anahtar Sözcükler: Uygulama Geliştirme, Sanallaştırma, 12 Faktör, Ruby on Rails.

Web Application Development Process Proposal Based on Container-based Virtualization and 12 Factors Methodology

Abstract: Nowadays, container-based virtualization has become popular. This is why there have been changes in the development process. By developers and system administrators, development and deployment processes are required to minimize their differences. And also differences between developers work environments are required to eliminate. Mentioned these problems, it has a critical role in the development process. In this study, modern solution proposals was made to these problems. Docker is used as container-based virtualization and Ruby on Rails web framework is used as web application.

Keywords: Application Development, Virtualization, 12 Factor, Ruby on Rails.

1. 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 SaaS uygulamaları inşa edebilmek için kullanılan bir yöntembilimdir. Bu yöntem ile birlikte şu sorunlara çözüm getirmektedir:

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.

Geliştirme ve üretim aşamaları arasındaki farklılıkları minimum seviyeye getirerek, hem konuşlandırma için kolaylık sağlar hem de çevik geliştirme modeli için sürekli dağıtım imkanı sağlar.

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

12 faktör yöntemliliği herhangi bir programlama dili ile yazılmış web uygulamasına uygulanabilir ve istenilen arka plan servisleriyle uyumlu bir şekilde çalışabilir (veritabanı, hafıza önbelleği, iş kuyruğu)

12 faktörü en iyi şekilde uygulamak için konteyner tabanlı sanallaştırma olan Docker biçilmiş kaftan niteliği görmektedir. Bu çalışmada da 12 faktörü Ruby on Rails web çatısı ile Docker üzerinde uygulamanın niteliklerine, ayrıntılarına ve faydalarına dair inceleme bulunmaktadır.

2. 12 Faktör Yöntemliliği

12 Faktör uygulaması SaaS uygulamaları inşa edebilmek için kullanılan bir yöntemliliğidir. Toplamda 12 ana unsura dayanmaktadır.

2.1 Kod Tabanı

12 faktör uygulamaları her zaman Git, Mercurial, Subversion gibi sürüm takip sistemleri ile izlenir. Kod deposu genellikle İngilizce terimlerinden dolayı repo veya Türkçe olarak kısaltılmak istendiğinde depo olarak kullanılmaktadır. Bu çalışmada depo olarak bahsedilmektedir.

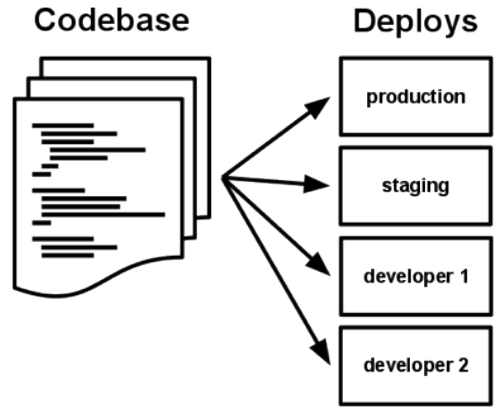
Kod tabanı ve uygulama arasında her zaman birebir ilişki vardır.

Eğer birden fazla kod tabanı varsa bu uygulama değil, dağıtık bir sistemdir. Dağıtık sistemlerde her kod tabanı bir uygulama ve bu uygulamaların her birine 12 faktör uygulanabilir.

Aynı kod tabanının birden fazla uygulamada kullanılması 12 faktör ihlalidir. Burada çözüm ortak kullanılan kodların bir kütüphane haline getirilerek, kullanılan programlama dilinin bağımlılık yöneticisi tarafından diğer sisteme entegre edilmesi sağlanmalıdır.

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

konuşlandırılabilir. Konuşlandırmak, uygulamanın örneğinin çalıştırılması anlamına gelmektedir. Basit olarak bir sitenin sunum ve bir veya birden fazla gösterim seviyelerinde çalıştırılmasıdır. Ayrıca her geliştirici kendi geliştirme ortamında uygulamanın kopyasını çalıştırabilir.



Şekil 1. Kod Tabanı

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

Örnek bir sürümlenme için komut satırında aşağıdakine benzer komut verilmelidir.

\$ docker build

<https://github.com/docker/rootfs.git#container:docker>

2.2 Bağımlılıklar

Pek çok programlama dili dağıtık destek kütüphaneleri için paketleme sistemi sunmaktadır; Bu çalışmada kullanılan Ruby on Rails için bu paketleme sistemi Bundle’dir. Gemfile adı verilen manifesto ile bağımlılıklar ortaya konur. İnşa komutu olarak da “bundle install” komutu kullanılır.

12 Faktör uygulamaları asla sistem genelinde kullanılan paketlere güvenerek onları kullanmamalıdır. Uygulamanın bağımlı

olduğu tüm bağımlılıklar doğru ve tam olarak bir manifesto ile deklare edilmelidir. Ayrıca bağımlılıkları ayrıştırma aracı ile sistem genelinden ayrıştırılmalıdır. Tüm bağımlılıklar geliştirme ve sunum aşamasında da aynı olmalıdır.

12 Faktör uygulamaların asla sistem geneli uygulamalara güvenmemeli ve onları kendi içlerinde kullanmamalıdır. Örneğin kabukta çalıştırılan yazılımlar olan “ImageMagick” veya “Curl”. Bu yazılımlar pek çok sistemde bulunmasına rağmen ileride bu yazılımların olacağının garantisi yoktur veya kullanılan versiyonun sizin yazılımınızla uyumlu çalışacağının garantisi yoktur. Eğer uygulamanız kabukta çalışacak bir yazılıma ihtiyaç duyuyorsa bu yazılımın sizin yazılımınız içerisine gömülü olarak hazır hale getirilmesi gerekmektedir.

Bu başlık altında bahsedilen konular 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.

2.3 Yapılandırma

Uygulama yapılandırması her konuşlandırma seviyesinde benzer adımlarla devam etmektedir. Bu adımlar:

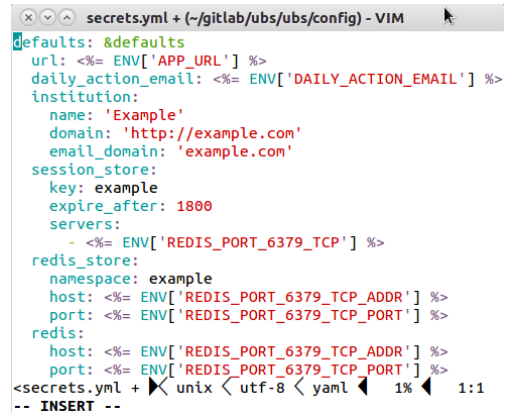
- Kaynak yönetimi (Veri tabanı, Önbellek mekanizmaları ve diğer geri plan işler)
- Amazon S3, Twitter gibi dış hizmetlerin hesap bilgilerinin yönetimi
- Her konuşlandırma için farklı bir isim seçilmesi

Uygulamalar bazen yapılandırmaları kod içerisinde saklamaktadır. Bu doğru bir yaklaşım değildir ve 12 faktör kurallarına aykırıdır. Yapılandırmaların kod içerisinde kesinlikle ayrılması gerekmektedir. Yapılandırmalar her konuşlandırmada önemli ölçüde

değişirken, kod değişmez.

12 Faktör uygulamaları yapılandırmaları çevre değişkenlerinde saklar (çoğu zaman çevre değişkenleri “env” şeklinde anılmaktadır). Çevre değişkenleri her konuşlandırmada koda hiç bir müdahale olmaksızın kolaylıkla değiştirilebilmektedir. Yapılandırma dosyaları gibi yanlışlıkla sürüm takip sistemlerine eklenme ihtimali azdır ve dil veya işletim sistemi standartlarına göre tanımlanmaktadır.

Bahsedilen “config/secrets.yml” dosyasından kısa bir alıntı olarak:



```
defaults: &defaults
url: <%= ENV['APP_URL'] %>
daily_action_email: <%= ENV['DAILY_ACTION_EMAIL'] %>
institution:
  name: 'Example'
  domain: 'http://example.com'
  email_domain: 'example.com'
session_store:
  key: example
  expire_after: 1800
servers:
  - <%= ENV['REDIS_PORT_6379_TCP'] %>
redis_store:
  namespace: example
  host: <%= ENV['REDIS_PORT_6379_TCP_ADDR'] %>
  port: <%= ENV['REDIS_PORT_6379_TCP_PORT'] %>
redis:
  host: <%= ENV['REDIS_PORT_6379_TCP_ADDR'] %>
  port: <%= ENV['REDIS_PORT_6379_TCP_PORT'] %>
<secrets.yml + |< unix < utf-8 < yaml | 1% | 1:1
-- INSERT --
```

Şekil 2. Ruby on Rails’de secrets.yml dosyası

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

docker-compose.yml dosyasında bu atama aşağıdaki gibi yapılabilir:

```
docker-compose.y...lab/ubs/ubs) - VIM
web:
  build: .
  command: bash -c "rm -f /app/tmp/pids/server.pid Gemfile.lock && bundle exec rails s -p 3000 -b 0.0.0.0"
  environment:
    DEFAULT_INSTITUTION_NAME: Example
    SECRET_KEY_BASE: 'asdfasdasfasasd'
    SYSTEM_EMAIL_USERNAME: test@example.com
    EMAIL_USERNAME: test@example.com
    EMAIL_PASSWORD: '*****'
    APP_URL: 'http://localhost:3000'
    DAILY_ACTION_EMAIL: false
  volumes:
    - ./app
    - ~/.pry_history:/root/.pry_history
    - ~/.mysql_history:/root/.mysql_history
  ports:
    - "3000:3000"
  links:
    - db
<| + | unix < utf-8 < yml ◀ 31% ◀ 13:27
-- INSERT --
```

Şekil 3. docker-compose.yml dosyası

2.4 Arka Plan Servisleri

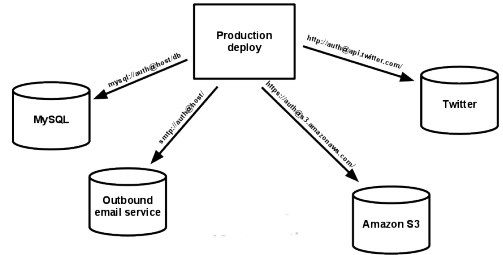
Arka plan servisi normal işlemleriyle ağı kullanan herhangi bir servis olabilir. Örnek olarak, veri tabanları (MySQL, CouchDB), mesaj ve iş kuyruğu sistemleri (RabbitMQ, Beanstalkd), elektronik posta gönderebilmek için kullanılan SMTP sistemleri (Postfix) ve ön bellek sistemleri (Memcached, Redis)

Arka plan servisleri, veri tabanları gibi geleneksel olarak sistem yöneticileri tarafından yönetilmektedir. Ek olarak aynı servisin hizmetlerini verecek üçüncü taraf uygulamalar da bulunmaktadır. Örnek olarak; SMTP servisi olarak Postmark, metrik toplama servisi olarak New Relic, dosya sunma servisi olarak Amazon S3 bunlara örnek gösterilebilir.

12 faktör uygulamaları yerel servisler ile üçüncü taraf servisler arasında ayırma yapmamaktadır. Her ikisinde de ulaşım URL veya servis kimlik bilgilerini yapılandırma ile olmaktadır. 12 faktör kodu, veri tabanı servisi değiştirildiğinde koda herhangi bir müdahale olmasını gerektirir. Yani servis farklılığı kodda değil yapılandırmada farklılık göstermektedir.

Uygulamaya eklenecek olan kaynaklar herhangi bir zamanda çıkarılabilir, eklenebilir olmalıdır. Örnek olarak veri tabanı sunucusunda meydana gelen bir arızada, arızalı donanımın çıkarılıp yenisinin takılması gerekmektedir. Bunu yaparken koda hiçbir şekilde müdahale olmamalıdır.

Docker-compose aracılığıyla arka plan servisleri sisteme kolayca entegre edilebilmektedir. Bu servisler sanki başka sunucudaymış gibi davranır. Web uygulamasının çalıştığı konteynerden farklı bir konteynerde çalışır. Aralarındaki bağlantı ilgili portların yalnızca birbirilerine açılması ile sağlanır.



Şekil 4. Arka Plan Servisleri

yanabilmek için sürekli yenilik içinde olmaları ve müşteri istek ve gereksinimlerini çok iyi bilerek müşteri odaklı üretim yapmaları zorunlu hale gelmiştir [10]. Özellikle ürün tasarımında kullanılan Kalite Fonksiyon Yayılımı (KFFY) gibi yaklaşımların müşteri odaklı olması son derece doğaldır [2].

2.5 İnşa Et, Duyur, Çalıştır

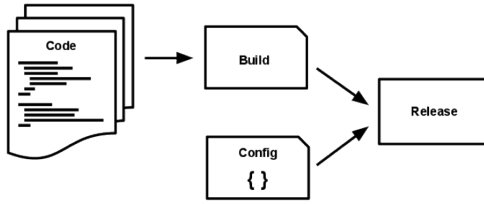
Kod tabanının konuşlandırılmasında (geliştirme süreçleri hariç) üç aşama bulunmaktadır.

İnşa etme aşaması: Bu aşamada kod tabanı inşa edilir. İnşa edilirken tüm bağımlılıklar ve gerekli olan yazılımlar sisteme dahil edilerek inşa edilmesi gerçekleştirilir. Ayrıca sistemin sunacağı varlıklar da sisteme dahil edilmektedir.

Duyuru aşaması: inşa etme sürecinin ardından uygulama içerisinde kullanılacak olan yapılandırmalar yapılır. Bu aşamadan sonra kod tabanı çalışma ortamında çalışabilecek durumdadır.

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

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.



Şekil 5. İnşa Et, Duyur, Çalıştır

2.6 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. 12 faktör uygulamalarında sunulacak olan ön bellek sistemi sürecin hafızadaki yerini kullanmamalıdır. Çünkü bir sonraki istekte aynı sürecin devam edeceğinin garantisi yoktur.

Ruby on Rails için örnek olarak Redis servisi kullanımı bellek durumunu kaydetme yardımcı olacaktır. Sistemde çalıştırılmak istenen her komut aynı redis kullanarak sistem durumunu paylaşmış sayılır.

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

2.7 Port Bağlama

12 faktör uygulamaları tamamen kendi içerisinde çalışan bir uygulamadır. Dışarıdan gelebilecek, web sunucuları üzerinden komut çalıştırma tarzı sızmalara karşı önlemi de budur. Bir port HTTP servisine tanımlanır ve bu uygulamada o porta gelen istekleri dinler.

Yerel geliştirme ortamında, geliştirici “<http://localhost:3000/>” gibi bir adrese erişebilmektedir. Konuşlandırmada ise yönlendirme katmanı istekleri açık ana bilgisayar adına göre ilgili porta yönlendirir.

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

ports:
- "3000:3000"

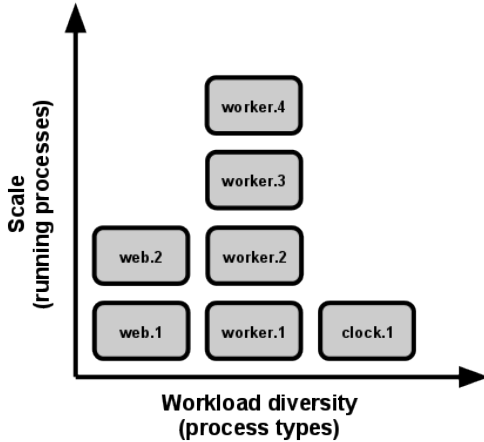
2.8 Eşzamanlılık

Bir bilgisayar programı çalıştığında, bir veya birden fazla süreç oluşturabilir. 12 faktör uygulamalarında süreçler birinci sınıf vatandaş gibidir. 12 faktör uygulamasındaki süreçler unix süreç modelinden ilham almaktadır. Bahsedilen bu tip model kullanılarak geliştirici uygulamada kullandığı mimariye göre işe tanımlarını dikkate alarak farklı süreç tipleri oluşturabilir. Buna 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.

Yukarıda bahsedilen durumu karşılamak için docker-compose biçilmiş kaftandır. Her bir servisi ayırmayı kolaylaştırmaktadır. docker-compose.yml dosyası içerisine worker'ların çalışması için yeni bir servis tanımlanabilir. Örnek olarak:

```
worker:
  build: .
  command: bundle exec sidekiq -e production
  -c 25
  environment:
    SECRET_KEY_BASE: asdasdasd...
    RAILS_ENV: production
  links:
    - db
    - redis
```

Şekil 6. docker-compose.yml worker servisi



Şekil 7. Süreç Tipleri

2.9 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 çalıştırılır ve çıkarılır. Bir daha bu konteyner kullanılmaz.

2.10 Geliştirme/Sunum Benzerliği

12 faktör uygulamaları geliştirme ve sunum aşamalarındaki farkı en aza indirgeyecek şekilde ve sürekli geliştirilecek biçiminde tasarlanmıştır. Yukarıda bahsedilen 3 farklılığa çözüm şu şekilde getirilmiştir.

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

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, sunucu katmanındaki farklılıkları gidermek için Docker, Dockerfile manifestosu sayesinde imaj derleme seçeneği sunulmaktadır.

2.11 Günlükler

12 faktör uygulamaları asla kendi çıktılarının yönlendirilmesiyle veya depolanmasıyla ilgilenmez. Günlük dosyalarına yazma ile veya bu dosyaların yönetilmesiyle ilgilenmemelidir. Mevcut geliştirme ortamında geliştiriciler uygulamanın çıktılarını çalıştırdıkları terminal üzerinden görecektirler.

Gösterim ve sunum gibi her aşamada çalışılan uygulamalar kendi ortamlarına göre günlük tutmalıdır ve arka planda çalıştırdıkları her

süreçten çıkacak olan çıktıları bir yönlendirici ile bir yerde saklamalıdır. Ancak bu saklama ve yönetim işini uygulamanın kendisi değil ayrı bir yönlendirici yazılımı ile yapılmalıdır. Bu amaçla ortaya çıkan açık kaynak pek çok yazılım mevcuttur (Logplex, Fluent).

Geliştirme ortamında halihazırda loglar proje dizinine log dizini altına kaydedilmektedir. Ayrıca docker-compose komutunun çalıştığı terminal ekranında ekrana dökmetedir. Hata ayıklama durumlarında bu çıktılar hayati önem taşımaktadır.

2.12 Yönetici İşlemleri

Süreç oluşumları uygulamaların çalışmasından ötürü sürekli oluşmaktadır. Örneğin web sunucularına gelen istekler bu süreçleri oluşturmaktadır. Bundan farklı olarak 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. Bu komutlar çıkarılacak olan sürümler üzerinde çalışır ve bunlar aynı kod tabanını paylaşan ve aynı yapılandırmaları paylaşan uygulamalar üzerinde çalıştırılır. Yönetici bu kodlar için oluşabilecek senkronizasyon problemlerinin farkında olmalıdır.

Bağımlılık izolasyonu tekniği süreçler için de kullanılmalıdır. Örneğin; veri tabanı göç işlemleri için “docker-compose run web bundle exec rake db:migrate” komutunu vermektedir.

3. Kaynak Kodlar

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

4. Sonuç ve Öneriler

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

Konteyner tabanlı sanallaştırma kullanarak uygulama geliştirme yöntemi başlık 2.2'de anlatılan geliştiriciler arasındaki geliştirme

ortamındaki bağımlılıkları bire bir yaptığından geliştirme ortamının hazırlanmasındaki vakit kaybını önlemektedir. Aynı şekilde yeni bir geliştiricinin yeni projeye başlaması da kolay hale gelmektedir.

Her ne kadar bu çalışma uygulama geliştirme süreçlerine odaklansa da uygulanan adımlar uygulamanın sunucu ortamında konuşturulmasını da kolaylaştırdığı açıktır. Sunucu ortamının docker aracılığıyla yerelde hazırlanıyor olması, olası sorunları önceden tespit etmeyi sağlamaktadır.

Docker ve 12 faktörün uygulanmasıyla hazırlanan uygulamaların sunucu ortamında ölçeklenmesi daha kolay olmaktadır. Bunun nedeni ise Arka plan servisleri ve yapılandırma kod tabanından bağımsız hale getirilmiştir. Geliştirme aşamasında docker'ı yönetmek için docker-compose kullanıldığı gibi konuşturma aşamasında da kullanılabilir.

Uygulamanın konteyner içerisinde çalışması geliştiricilerin sunucuda hesap istemelerinin de önüne geçecektir. Çünkü uygulamanın çalıştığı konteynere ssh veya benzeri yöntemlerle bağlanmaya izin verilmez. Bunun yerine 2.9'da anlatılan tek kullanımlık ilkesi uygulanır.

5. Kaynaklar

[1] Docker Container - <https://docs.docker.com/userguide/usingdocker/>. (Ziyaret Tarihi: 10 Ağustos 2015)

[2] Docker Compose - <https://docs.docker.com/compose/> (Ziyaret Tarihi: 11 Ağustos 2015)

[3] Ruby on Rails - <http://rubyonrails.org/> (Ziyaret Tarihi: 9 Ağustos 2015)

[4] The Twelve-Factor App - <http://12factor.net/> (Ziyaret Tarihi: 9 Ağustos 2015)