Mevcut Bir Bulut Uygulamanın Docker ve Docker Teknolojilerini Kullanarak Bağımsız Çalışabilir Hale Getirilmesi Deneyimi (Experience of Converting an Existing Cloud Application into a Standalone System by Using Docker and Related Technologies)

Serdar Mumcu[1][0000-0002-4919-0015]

[1] Comodo Threat Research Labs, NuRD Inovasyon Merkezi, Ankara
serdar.mumcu@nurd.com

Abstract. In this paper, some basic information about Docker technology, which is frequently mentioned nowadays, is given and its differences with and advantages over virtualization technologies are mentioned. In addition to this, a brief information about Valkyrie architecture, which is a cloud-based product, is given and the actions to be taken to meet the requirements and expected problems are mentioned. Finally, the experience of converting this product into a standalone package using Docker and Docker technologies; the way Docker and Docker Swarm technologies are utilized throughout this experience; and additional benefits obtained while using this architecture have been discussed.

Özet. Bu bildiride temel olarak öncelikle günümüzde adından sıkça bahsettiren Docker teknolojisi hakkında temel bilgiler verilmiş, sanallaştırma teknolojileri ile olan farklılıklarından ve avantajlarından söz edilmiş, akabinde mevcut bir bulut (cloud) ürün olan Valkyrie mimarisi ile ilgili kısa bir bilgilendirme ve çözülmesi gereken ihtiyacın ve karşılaşılabilecek olası sorunların aktarılması sonrasında Docker ve Docker teknolojilerini kullanarak bahsedilen mevcut bulut ürünü bağımsız (standalone) bir kurulum haline getirme deneyimine ve bu deneyim sırasında Docker ve Docker Swarm teknolojilerinin konumlandırılma yöntemine ve ilgili mimarileri kullanmanın getirdiği ek faydalara değinilmiştir.

Anahtar Kelimeler: Standalone, On-premise, Docker, Docker Swarm, Valkyrie, Dockerfile, Docker-compose, Overlay Network.

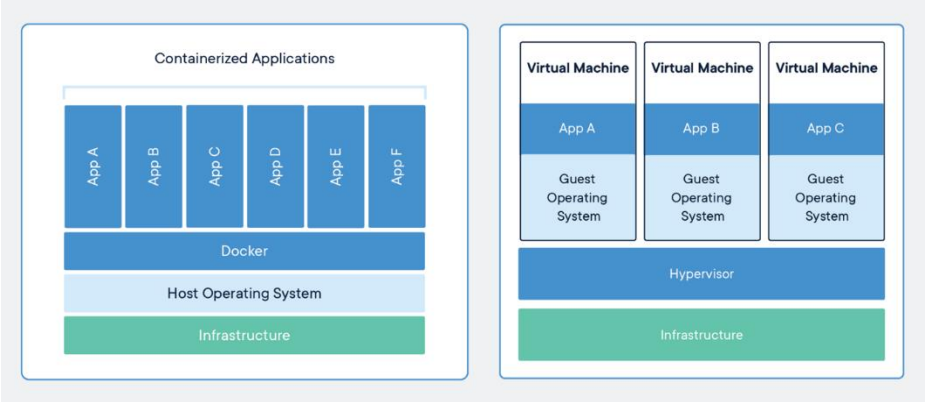
1 Giriş ve Docker Hakkında

Günümüzde oldukça popüler bir hale gelen Docker teknolojisi aslında LXC [1] dediğimiz (Linux Container) IBM tarafından 2008 yılında Linux çekirdeğine eklenmiş, işletim sistemi seviyesinde sanallaştırma sağlayan bir teknolojiyi temel alarak geliştirilmiştir. Özellikle uygulamaları aynı ev sahibi işletim sistemi (Host Operating System) üzerinde birbirinden izole şekilde çalıştırabilmesiyle öne çıkmış

bir teknolojidir. Bunu yaparken de sıklıkla kullandığımız sanal makinelerden oldukça farklı bir yöntem kullanmaktadır [2].

Öncelikle sanal makineler tam anlamıyla tüm işletim sistemi modüllerine sahip bir misafir işletim sistemi (Guest Operation System) çalıştırmak zorunluluğu ile gelirler. Bu işletim sisteminin açılması, yeniden başlatılması gerekli modüllerin yüklenmesi, servislerinin ayağa kalkması oldukça ciddi bir zaman alır. Aynı zamanda tüm bu ek işletim sistemi modül, servis ve sürücüleri belli bir oranda üzerinde çalıştığı donanımın işlemci, bellek ve disk kaynaklarından fazladan tüketime neden olurlar. Bu kaynaklara getirilen ek tüketim aslında ek maliyet anlamına da gelir [3].

Sanallaştırmanın temel amacı mevcut sahip olduğumuz donanımlar üzerinde bağımsız uygulamalar çalıştırmaktır. Docker bu anlamda ihtiyaç duyulan açığı kapatan bir teknoloji olarak karşımıza çıkmaktadır. Docker içinde bulunduğu işletim sisteminin (Host OS) dışında ek bir işletim sistemine ihtiyaç duymadan uygulamalarımızı birbirinden izole çalıştırabilmemizi sağlar.



Şekil 1. Docker ve Sanal Makine Farkı [12]

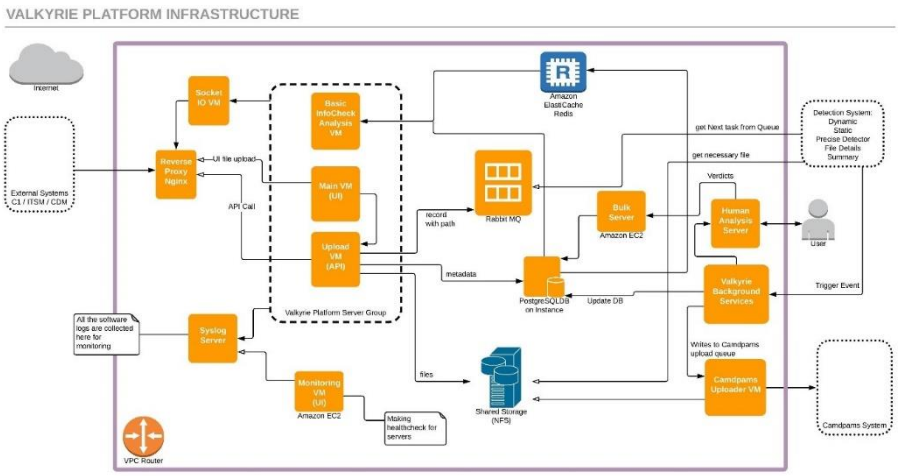
Bunu yaparken de oldukça hızlı ve donanımımıza da ek bir kaynak tüketimi getirmeden bu işlemleri gerçekleştirebilmemize yardımcı olur. Her uygulama aynı işletim sistemi üzerinde birbirinden bağımsız olarak kendi bağımlılıklarıyla çalışabilir ve uygulamalar kolaylıkla olduğu haliyle başka bir ortama taşınabilir. Özellikle geliştiricilerin yakındığı ve kendi makinelerinde çalışan kodun ya da uygulamanın sunucuda çalışmaması şeklindeki sorunların oluşmasının en başından önüne geçer [4].

Aslında Docker geliştiricinin geliştirme şeklini ve uygulamalarımızı dağıtma yöntemlerimizi temelinden değiştirerek bizi mikroservis (microservice) mimarisine de hazır hale getirir [5].

2 Ürün ve İhtiyaç Hakkında

Bu bildiride, NuRD Inovasyon Merkezi altında faaliyet gösteren Comodo Threat Research Labs birimi olarak gerçek bir ihtiyacımızı Docker'ın getirdiği olanaklar ile çözdüğümüz ve sonuçlarından oldukça memnun kaldığımız bir senaryomuz ve bu senaryo sırasındaki elde ettiğimiz deneyimlerimiz ve edindiğimiz faydalar paylaşılmaktadır. Firmamızda geliştirmekte olduğumuz Valkyrie: Gerçek Zamanlı Tehdit Tespit Sistemi (Valkyrie: Real Time Threat Identification Service) isimli sistemimiz aslında bulut üzerinde sunduğumuz birbirinden çok farklı bileşenlerden oluşan bir altyapıya sahip bir üründür. Ürün içerisindeki bileşenler uygulama sunucuları (web application servers), veri tabanı sistemleri (database systems), kuyruk mekanizmaları (queue systems), hafızalama sunucuları (cache) vb. gibi çoğu açık kaynaklı ürünlerdir. Normalde, Valkyrie ürününü bir bulut sistem hizmet sunucusu (cloud provider) üzerinden hizmet alarak hizmete sunulmaktadır ancak bazı büyük kurumlar bu sistemi güvenlik ve bilgi gizliliği gibi nedenlerle kendi ortamları içerisinde bağımsız (standalone) kurulum şeklinde kullanmak isteyebilmektedir.

Sadece bellekleme sunucusunun (cache server) bile 7 farklı sunucudan oluşan bir yapıyı kurumun kendi ortamına sadece kurmak ve konfigüre etmek bile başlı başına büyük bir sorun iken bir de ilgili kuruma en az yerinde destek ile sistemin bakım ve idamesini gerçekleştirmek büyük zorluklar getirmekte ve karmaşa yaratmaktadır.



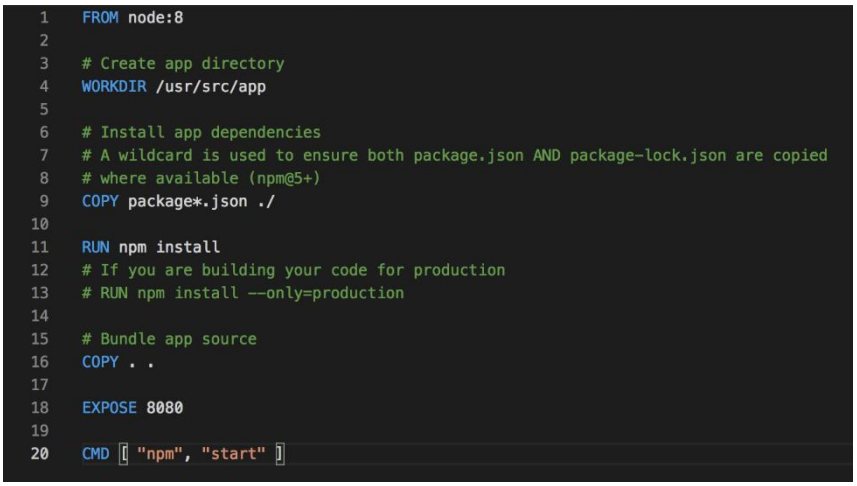
Şekil 2. Valkyrie Mevcut Mimari

3 Uygulanan Çözüm Hakkında

Bu soruna çözüm olarak sistemdeki tüm uygulamaları ve bileşenleri öncelikle dockerize etmek ve ilgili kurumun bu çözüm için tahsis ettiği makine sayısı ne olursa

olsun, oluşturulacak docker container'ları ilgili sunuculara yine Docker'ın sağladığı Docker Compose ve Docker Swarm teknolojilerini kullanarak dağıtmak şeklinde bir çözüm benimsenmiştir.

Docker'laştırmak (dockerization) aynı zamanda bu bileşenlerin kurulumlarını standart hale getirmek ve dokümante etmek anlamına da gelmektedir. Bunun için Dockerfile ismini verdiğimiz özel bir dosya oluşturulmaktadır [6].



```
FROM node:8

# Create app directory
WORKDIR /usr/src/app

# Install app dependencies
# A wildcard is used to ensure both package.json AND package-lock.json are copied
# where available (npm@5+)
COPY package*.json ./

RUN npm install
# If you are building your code for production
# RUN npm install --only=production

# Bundle app source
COPY . .

EXPOSE 8080

CMD [ "npm", "start" ]
```

Şekil. 3. Dockerfile Örnek Dosya

Bu dosyalar da yine uygulama kodları gibi bir kaynak kontrol (source control) sistemi üzerinde saklanmaktadır. İstenildiği ya da gerektiğinde bu dosyalar güncellenerek kurulum parametreleri değiştirilebilmektedir. Daha öncesinde de her ne kadar kurulum adımları dokümante edilse de (operasyon dokümanları - runbook'lar ile) her seferinde bu işlemlerin bir kişi tarafından baştan sona adım adım uygulanması gerekmekteydi. Aynı zamanda bazen manuel yapılan çözümlerin bu dokümantasyona yansıtılmaması gibi sorunlar da oluşabilmekteydi. Bu çözümü gerçekleştirebilmek için öncelikle tüm bu farklı bileşenleri geliştirici makinesinde çalışabilir hale getirilmesi hedeflenmiştir. Normalde geliştirici makinesine tüm bu bileşenler kurulabilse bile bazıları küme yapıda olduğu için birden fazla makineye ihtiyaç duyuluyor ve gerçek ortam ile eşlenik bir kurulum yapılamamaktaydı.

Öncelikle bu sorunu çözebilmek için tüm uygulama ve bileşenler docker'laştırılmıştır. (Nginx-uwsgi, Redis, RabbitMQ, ElasticSearch, Logstash, SocketIO, Syslog, Jenkins) Dockerlaştırma sırasında kalıcı (persistent) olması gereken veriler için Docker'ın takılan birim (Volume Mount) [14] özelliğinden faydalanılmıştır. Daha sonra tüm dockerize edilen ilgili bileşenler Docker Compose ürünü ile tanımlanmıştır. Bu ürün sayesinde geliştirici sadece tek bir komut çalıştırarak bütün sistemi kendi makinesinde docker container'lar içerisinde ve sanal bir ağ ile çalışabilir hale getirebilmektedir. Bu sayede onlarca sanal makine

kullanarak kuralabilen altyapıyı tek bir makine üzerinde de hem çok hızlı hem de çok kolay ayağa kaldırabilme olanağı sağlamıştır. Yine docker-compose için de docker-compose.yml adını verdiğimiz özel bir dosya türü kullanılmaktadır. Özellikle docker-compose ürünü, yeni bir geliştirici ekibe dahil olduğunda geliştirme ortamının yeni kişiye sorunsuz şekilde kurulması amacı ile de kullanılabilmektedir [7].

```yaml
version: '3'

services:
  product-service:
    build: ./product
    volumes:
      - ./product:/usr/src/app
    ports:
      - 8080:5000

  website:
    image: php:apache
    volumes:
      - ./website:/var/www/html
    ports:
      - 5000:80
    depends_on:
      - product-service
```

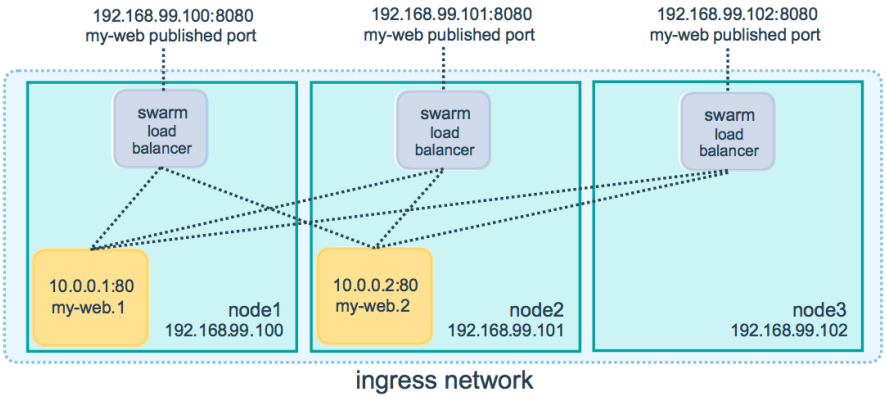
Şekil. 4. Docker-compose.yml Örnek Dosya

Standalone kurulum oluşturma işleminin son aşamasında ise geliştiricinin tamamen çalışabilir hale getirdiği bu sistemi bir konteynır orkestratör (container orchestrator) yardımıyla birden fazla makineden oluşan bir gruba kurup çalıştırılması hedeflenmiştir. Bunun için de oldukça kullanışlı olan bir orkestratör olan Docker Swarm teknolojisinin kullanılması tercih edilmiştir. Docker Swarm'ün öncelikle mevcut makinelere tanımlanması sağlanmıştır. Demo ortamımızda 3 adet Linux makine oluşturulmuştur. Bu makineler yine Docker'ın sağladığı docker-machine scriptleriyle oluşturulduğundan herhangi bir ek kurulum yapmadan direkt olarak docker swarm kurulumuna geçilebilmiştir. Docker swarm kurulumu gerçekleştirildikten sonra mevcut docker-compose.yml dosyası, ona çok yakın bir biçim olan docker-stack dosyasına dönüştürülmüştür.

Kurulumu yapılan makinelerden herhangi birinde bu stack dosyasını çalıştırarak uygulamaların kurulumu (deployment) gerçekleştirilmiştir. Tüm servis ve bileşenler 3 makineye docker swarm tarafından dağıtılarak kurulumları gerçekleştirildi. Bu

aşamada gelen istekleri (request) bu makinelere dağıtan bir yük dengeleme (Load Balancer) ayarı yapılmıştır. Docker Swarm Orchestrator ilgili servis başka bir makinede bile olsa bu isteği alıp ilgili servise arka planda bir üst katman sanal ağ (overlay/ingress network) üzerinden gönderip cevabını alıp bize gönderme yeteneğine sahiptir. Dolayısıyla servisin hangi makinede çalışır durumda olduğu bizim için bir önem arzetmemektedir [8] (Service discovery).

Bunların dışında yine Docker Swarm sağlamış olduğu sağlık kontrolü (healthcheck) ve yeniden başlama (auto-restart) özellikleri sayesinde herhangi bir servis işlev dışı kaldığında bunu otomatik olarak algılayarak servisi yeniden başlatmaktadır. Bu özellikler sayesinde standalone kurulum yaptığımız kurum sistemi üzerindeki bakım maliyetleri de minimize edilmiş oldu [9]. Aynı zamanda Docker swarm kolay ölçeklendirme özellikleriyle uygulamanın ölçeklendirilmesinin rahatça gerçekleştirilmesi [10] (Auto Scaling), belirli sayıda servisin sürekli çalışır durumda kalmasını sağlaması (Desired state reconciliation) ve bileşenlerin güncellemelerinin sistemde herhangi bir durma olmadan sağlaması (Zero Downtime Deployment) gibi ek özellikler ile de fayda sağlamıştır [11]. Uygulama içerisinde bir bileşene ait çalışan konteynır (Container) sayısını belli bir seviyenin altına düşürmek istenmediğinde veya ilgili bileşene anlık olarak çok fazla istek (request) geldiğinde sadece o bileşenin çalışan konteynır (container) sayını otomatik olarak arttırmak istediğimizde ya da yeni bir sürüm yüklediğimizde (deploy) hizmet kesintisi olmaksızın istekleri yeni sürüme taşımak istediğimizde Docker Swarm teknolojisinin getirdiği bu özellikler sayesinde bu işlemler rahatlıkla gerçekleştirilebilmektedir.



Şekil. 5. Docker Swarm Altyapısı [13]

4 Sonuç ve Değerlendirmeler

Docker ve Docker Teknolojilerini kullanarak oldukça zor olan bağımsız (standalone) kurulum işlemlerini oldukça basite indirgenmesi ve standart hale getirilmesi sağlanmıştır. Aynı zamanda bakım maliyetleri düşürülmüş, projenin bakım ve idame

ihtiyaçları da minimize edilmiştir. Bu şekilde mevcut bulut ortamlarımının zorlanmadan bağımsız (standalone) halinde sunulabilmesi aynı zamanda şirketimize yeni ve önemli bir iş sahasının kapılarını açmıştır.

Sonuç olarak standalone kurulum problemine çözüm sunmak amacıyla kullanılan Docker ve Docker teknolojileri aynı zamanda aşağıdaki ek faydaları getirmiştir:

- Kaynak kullanımı verimliliği (Hardware Utilization): İlk bölümde bahsedilen [3] nedenlerle Şekil2 ve Şekil5'te de göründüğü gibi 15 makine'den oluşan bir sistem, bileşen sayısı, mimarisi ve performans değişikliği olmadan 3 makinede çalışabilir hale indirgenebilmiştir.
- Çalıştırılabilen bir dokümantasyon oluşturabilme (Infrastructure as Code): 3. Bölümde bahsedilen Dockerfile ve Docker-Compose.yml dosyalarıyla systemin tüm bileşenleri hem tek bir dosyada tanımlanmış ve dokümante edilmiştir. Hem de bu dokümantasyon istenildiği takdirde geliştirici bilgisayarında veya sunucuda çalıştırılabilmektedir.
- Kolay kurulum (Ease of Deployment): Docker Swarm bileşeni sayesinde kurulum ortamı kaç sunucudan oluşursa oluşsun bunlar bir kümeye (cluster)'a dahil edilebilmekte ve tüm altyapı tek bir makine gibi davranmaktadır. Yeni bir sürümü yüklemek için sadece tek bir dosyayı değiştirmek ve tek bir komut çalıştırmak yeterli olmaktadır [15].
- Bakım ve yönetim kolaylığı (Maintainability): Docker Swarm'da kümeyi (cluster) yönetmek, izlemek, makinelerdeki hata durumlarına karşı önlem almak için makinelerin köle-efendi (master-slave) yetkilendirmesini ayarlamak gibi bir çok işlem çok hızlı ve kolay bir şekilde gerçekleştirilebilmektedir. Gerektiğinde yazılım geliştirici bile kendisi kümeyi yönetebilmektedir [16].
- Sağlık Kontrolü ve Otomatik Yeniden Başlatma (Healthcheck and Auto-Restart): Docker Swarm kapanan konteynırları farkedip kendisi yeniden başlatabilmektedir. Bunun için sağlık kontrolü (Healthcheck) mekanizmasına neyi kontrol edeceği tanımlanabilmektedir [9].
- Otomatik Ölçeklendirme (Auto Scaling): Zaman zaman sistem bileşenlerindeki bir takım alt servislerin yükü diğer servislere oranla daha fazla artabilmektedir. Bu gibi durumlarda yazılacak bir takım basit betikler (script) ile Docker Swarm'ın bu bileşenleri otomatik olarak ilgili servislerin çalışan ortam sayısını yüke ya da izlenen bir kuyruk sistemindeki bekleyen iş sayısına göre arttırıp azaltılması sağlanabilmektedir. Yine istediği takdirde kümeye yeni makineler de otomatik olarak eklenebilmektedir. Bu durum hizmet kalitesinin düşmemesini sağladığı gibi makine sayıları optimal tutulduğu için aynı zamanda maliyet açısından da fayda sağlamaktadır [10].
- İstenen Durum Uyumlaştırma (Desired State Reconciliation): Docker Swarm içerisindeki mekanizmalar sayesinde her zaman hedeflenen durumu gerçekleştirmeye çalışır. Bunun için anlık durumu sürekli kontrol ederek hedeflenen durumun dışına çıkıldığında bu duruma tekrar ulaşılması için gerekli işlemleri kendisi otomatik olarak gerçekleştirir. Örneğin, bir servis'in çalışan konteynır sayısı 3 olarak ayarlanmış ancak anlık çalışan servis sayısı 2 olmuş ise

sistem otomatik olarak yeni bir konteynır ayağa kaldırır. Böylece hedeflenen duruma ulaşmış olur [17].

- Hizmet Kesintisiz Uygulama Güncelleme (Zero Downtime Deployment): Docker Swarm üzerinde bir uygulamanın yeni bir sürümü kurulmak istendiğinde herhangi bir hizmet kesintisi olmadan bu kurulum hızlı bir şekilde gerçekleştirilebilmektedir. Kullancılar sürüm 1'i kullanırken kurulum sürüm 2 kurulduğunda sistem 2. sürüm istekleri kabul edebilene kadar istekleri 1.sürüm'e iletmeye devam eder. 2.sürüm ayağa kalktığında ise 1.sürüm'e giden istekleri yavaş yavaş 2. Süreme aktarmaya başlar. Daha sonra tümü aktarıldığında 1.sürümü devre dışı bırakır [18].
- Hızlı geliştirme ortamı kurulumu (Faster Local Dev. Environment Creation): Geliştirme ekibine yeni biri katıldığında onlarca bileşeni kurmak ve konfigüre etmek yerine sadece tek bir komut çalıştırarak tüm ortamı bilgisayarında ayağa kaldırabilmektedir. Hızlı bir şekilde geliştirme süreçlerindeki ihtiyaçlara odaklanabilmektedir [19].

Bu bağlamda edinilen bu deneyimlerden ve kazanımlardan benzer projelerde faydalanılması planlanmaktadır.

5 Teşekkür

Bildirinin hazırlanması sırasında gerek içerik gerek dil yönünden yaptıkları önemli katkı ve kritikler ile desteklerini esirgemeyen değerli çalışma arkadaşlarım ve yöneticilerim Gürkan Karahan, Anıl Doğan, Özgür Devrim Orman, Nurettin Mert Aydın, Can Peker, Uğur Çakır ve Fatih Orhan'a desteklerinden ötürü teşekkür ederim.

Referanslar

1. LXC Wikiwand, https://www.wikiwand.com/en/LXC, last accessed 2018/11/07
2. Containers vs. Virtual Machines (VMs): What's the Difference?, https://blog.netapp.com/blogs/containers-vs-vms/, last accessed 2018/11/07
3. Evaluation of Docker containers based on hardware utilization, https://ieeexplore.ieee.org/document/7432984/, last accessed 2018/11/07 (Preeth E N, F. J. P. Mulerickal, B. Paul and Y. Sastri, "Evaluation of Docker containers based on hardware utilization," 2015 International Conference on Control Communication & Computing India (ICCC), Trivandrum, 2015, pp. 697-700. doi: 10.1109/ICCC.2015.7432984)
4. But, it works on my machine. - Hacker Noon, https://hackernoon.com/but-it-works-on-my-machine-74b6875ab4e7, last accessed 2018/11/07
5. Advantages of Using Docker for Microservices, https://rubygarage.org/blog/advantages-of-using-docker-for-microservices, last accessed 2018/11/07
6. Infrastructure as Code: A Reason to Smile | ThoughtWorks, https://www.thoughtworks.com/insights/blog/infrastructure-code-reason-smile, last accessed 2018/11/07

7. Using Docker Compose for Development Environment Dependencies, https://vsupalov.com/flask-docker-compose-development-dependencies/, last accessed 2018/11/07
8. Service discovery with Docker Swarm, https://blog.scottlogic.com/2016/06/17/docker-swarm.html, last accessed 2018/11/07
9. Docker health checks - Dots and Brackets: Code Blog, https://codeblog.dotsandbrackets.com/docker-health-check/, last accessed 2018/11/07
10. Auto-scaling a Docker Swarm, https://scene-si.org/2017/05/02/auto-scaling-a-docker-swarm/, last accessed 2018/11/07
11. Swarm mode overview | Docker Documentation, https://docs.docker.com/engine/swarm/, last accessed 2018/11/07
12. What is a Container | Docker, https://www.docker.com/resources/what-container#/package_software, last accessed 2018/11/07
13. Use swarm mode routing mesh | Docker Documentation, https://docs.docker.com/engine/swarm/ingress/#publish-a-port-for-tcp-only-or-udp-only, last accessed 2018/11/07
14. Use volumes | Docker Documentation, https://docs.docker.com/storage/volumes/#choose-the--v-or---mount-flag, last accessed 2018/11/07
15. Viktor Farcic, The DevOps 2.1 ToolKit: Docker Swarm, pp. 159 – 165. Packt Publishing, Birmingham (2017)
16. Administer and maintain a swarm of Docker Engines, https://docs.docker.com/engine/swarm/admin_guide/, last accessed 2018/11/07
17. Viktor Farcic, The DevOps 2.1 ToolKit: Docker Swarm, pp. 58 – 62. Packt Publishing, Birmingham (2017)
18. Srdjan Grubor, Deployment with Docker, pp. 272 – 277, Pact Publishing, Birmingham – Mumbai (2017)
19. How Docker can improve your development environmentsi https://medium.com/phytochemia-tech-blog/how-docker-can-improve-your-development-environments-731cdfda0b9a, last accessed 2018/11/07