BLM 6110

PARALEL VE DAĞITIK HESAPLAMA DERSİ

EROL ÇITAK

18501033

1- GİRİŞ

Günümüzde görüntü işleme ve makine öğrenimi tekniklerinde kat edilen yol ile beraber kazandığımız yeteneklerin kapasitesi artmıştır. Gerek cep telefonlarındaki grafik işleme birimlerinin kullanımı gerekse de büyük sunucularda yapılan ve son kullanıcıya sunulan işlemler arasında şunları söyleyebiliriz;

- İmge/Video Altyazılama
- Nesne Tanıma
- Yarı Otonom Sürücüsüz Araç- Seviye 2-4
- Röntgen Filmlerinin Analizi

gibi birçok teknolojik gelişme görüntü işleme, bilgisayarla görü ve makine öğrenimi alanlarının bileşimi ile olmaktadır. Öyle ki, yukarıda örnek gösterilen çalışmaların yanı sıra yapay zekanın yapmış olduğu bir portre 432 bin dolarlık satış fiyatı [1] ile insanoğlunun varoluşundan bu yana kendisine atfetmekten alıkoyamadığı sanatsal düşünceyi sorgular hale gelmiştir.

Bu gelişmelerin yanı sıra; yapay zekâ teknolojilerinin karar verme, daha önce almış olduğu kararlarından yeni bir durum için karar oluşturabilme gibi yeteneklerinin ise şu an çok kısıtlı olduğu yine bilinen bir gerçektedir. [2]

Biz bu ders kapsamında ise bu teknolojilere ön ayak olmuş ve halen de bu çalışmalar sırasında ön-işlem adımları olarak kullanılan 5 farklı servisi ele aldık. Bu servisler;

- Renkli resmin (RGB kanallı) gri tonlamalı bir resim haline çevrilmesi
- Gri tonlamalı bir resim üzerinde kenar hatlarının bulunması
- Gri tonlamalı bir resmin kenar hatlarının keskinleştirilmesi
- Gri tonlamalı bir resmin kenar hatlarının yumuşatılması
- Gri tonlamalı bir resim üzerinde göze çarpan (salient points) noktaların tespiti

Örnek vermek gerekirse; göze çarpan noktalar, insanlardaki görme sistemi olan fovea sistemini taklit etmeye çalışarak; girdi olarak verilen bir imge ve/veya video çerçevesi üzerinde önemli bilgi içerebilecek noktaları tespit etmektedir. Bu sistemin kullanım alanları olarak, daha önce gerçekleştiğim bir uygulamayı verebilirim. 4 kanatlı bir helikopter üzerinde bulunan bir kamera ile nesne tespiti uygulamamdaki kısıt, alınan yer görüntüsünün üzerindeki nesneleri yerdeki sunucuya göndermeden anında helikopter üzerinde yapmaktı.

Fakat bu tespit işleminin yüksek başarımla yapılabilmesi için gerekli olan donanım ve enerji gereksinimlerinin varolmamasından ötürü, alınan video çerçevesi girdisine öncelikle göze çarpan noktalar tespiti yapmış, bu algoritmanın çıktısı üzerinde daha az nokta üzerinde nesne tespiti algoritmaları çalıştırılmıştır. Bu sayede yakın-gerçek zamanlı bir uygulama, daha az sistem gereksinimleri üzerinde çalıştırılabilmiştir.

Bu kapsamda yukarıda belirtmiş olduğum algoritmaları bir web sunucusu üzerinde, mikroservis mimarisinde gerçekleştirdim. Burada mikroservis mimarisini kullanmamın sebebi; bahsetmiş olduğum 5 farklı algoritmadan bir ve/veya birkaçının bağlı bulunduğu altyapılarda sorun olması durumunda bile diğer algoritmaların sağlıklı bir şekilde çalışmasını sağlayabilmektir. Bunun yanı sıra ileride doğabilecek ihtiyaçlar için tüm uygulamayı değiştirmektense bu birimleri tek tek değiştirmenin getireceği kolaylıktır.

Bu bilgiler ışığında, uygulamamı Python dili ile, Flask web sunucusunu kullanarak inşa etmeye başladım. Kullanıcıdan resmin girdi olarak alınması ve hangi operasyonların uygulanacağına dair komutların alınması için gerekli altyapıyı ise Knockout.js ile gerçekleştirdim. Docker kullanarak her bir servis için ayrı ayrı bir imge ve ondan da konteynır oluşturarak, bunlar arasındaki bağlantıyı öncelikle "docker-compose.yml" da belirttim ve sorgularımı oluşturmuş oldum.

2- SİSTEM MİMARİSİ, KULLANIM SENARYOLARI, TEKNİK ZORLUKLAR

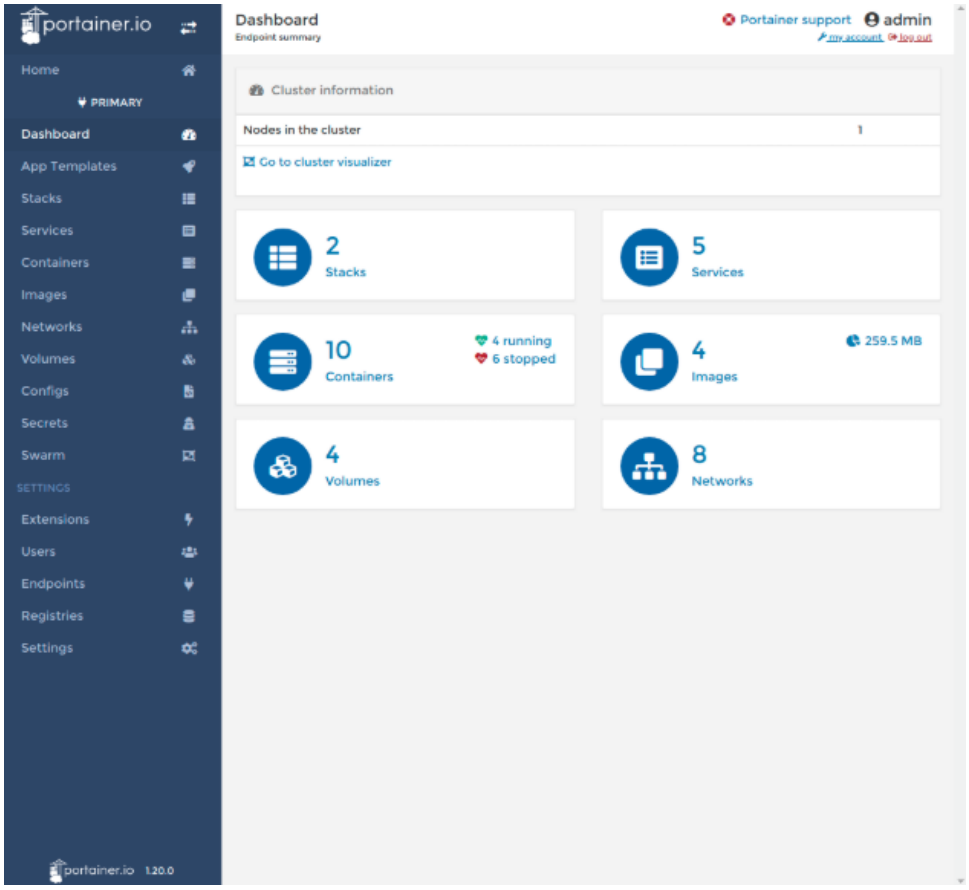
Sistemimiz 5 adet servis ve bir adet kullanıcı ekranına sahip ana servisten oluşmaktadır. Kullanıcı ana servis üzerinden 5 farklı servise sunabileceği resmi seçmektedir. Daha sonra bu resim, servisimiz üzerinde kayıt altına alınmaktadır. Bu imgeyi, kullanmak istediği servise gönderebilmekte ve sonucunu gönderdiği servisin altında kayıt altına alabilmektedir.

- Kullanıcı renkli ve/veya gri tonlamalı imgesini girdi olarak sisteme ekleyebilir veya daha önce eklediği imgeyi kaldırabilir.
- Girdi imgesi üzerinde; 5 farklı operasyon uygulayabilir ve bunların sonuçlarını gözlemleyebilir.

Bu proje kapsamında bir web sunucusu oluşturmak, bir uygulamayı Docker ile sarmalamak ve birden fazla Docker konteynırını konuşturmak ayrıca ön yüz için kullanılacak Knockout.js ve onun sunmuş olduğu MVVM yapısını anlamak ve proje üzerinde gerçeklemek benim için zorlayıcı oldu. Çalıştığım şirketteki pozisyonum gereği bu işlem adımlarına hiç girmeden sadece akademik başarımı hedeflediğimizden bu teknolojileri bir arada kullanmak benim için bir ilkti ve öğretici oldu.

Bu zorluklar kapsamında özellikle; birden fazla Docker servisini konuşturmak için, birden fazla olan network yapısını denemek ve sonucunda docker-compose.yml dosyasından bu servisleri ana servisimize takma isimleri(alias) ile vermenin erişmenin benim için en uygun olduğunu buldum.

Docker imgeleri ile çalışırken, bir servis için birden fazla konteynır oluşturulması, bu konteynırların yönetimi gibi konularda bazı zorluklar yaşadım. Bunları aşabilmek için, bu konteynırlara ve imgelere bunların yüklerine erişebileceğim bir modüle ihtiyacım doğdu. Bu kapsamda Portainer [3] isimli bir uygulamaya denk geldim. Portainer sayesinde anlık olarak hangi imge ve konteynırların aktif olduğunu, üzerlerindeki yükleri ve birçok ekleme, kopyalama, silme gibi operasyonları hızlı bir biçimde gerçekleştirme imkânı buldum.



Figür 1: Portainer.io kullancı arayüzü

3- UYGULAMANIN GERÇEKLENMESİ

Uygulamam içerisinde web sunucu olarak, Flask çatısını tercih ettim. Bu kapsamda kolay ve hızlı bir biçimde uygulamamın iskeletini ayağa kaldırabilmiş oldum. Bu web sunucuna vermiş olduğum port numarası ile sunucu üzerindeki debug kapılarını açık tutarak anlık olarak alınan ve gönderilen isteklerin durumlarını analiz edebildim.

Docker imgesi oluşturma ve bunlardan konteynırların oluşturulması işlemlerini her bir servis için ayrı ayrı yaptım. Ve bunların oluşturulma sürecinde öncelikle python-3 development altyapısını kullanan imge tabanı üzerine; flask, requests, opencv, numpy kütüphanelerini kullandım. Daha sonrasında bu konteynırları birbirleri ile konuşturabilmek adına, docker-compose.yml içerisinde bu servisleri birbirine *links* yapısı ile bağladım. Bu sırada docker konteynırlarına bağlanmak için ve konteynır içerisinde web sunucularına ve servislerine bağlanacak portları da belirttim.

Görüntü işleme kısmında open-source bir kütüphane olan Opencv'yi kullandım. Opencv temel ve gelişmiş birçok operasyonu bünyesinde barındıran ve yaklaşık 6 ayda bir kendini güncelleyen bir open-source kütüphanedir.

Bu sırada projemiz kapsamında gerekli olmamasına karşın; Knockout.js kütüphanesi ile kullanıcıdan komut girdisi alabilecek bir önyüz buluşturucusunu kullandım. Bu yapı **M**odel**V**iew**V**iew**M**odel yapısını kullanarak, girdi komutlarını hızlı bir biçimde istemciye sunabilmektedir.

4- SONUÇ VE GELECEK ÇALIŞMALAR

Bu proje kapsamında; mikroservis mimarisi, docker yapısı ve web sunucularının iletişimi konularında öncül bir bilgi edinmiş bulundum. Bunun yanı sıra üçüncü parti yazılımlar ile yerelde tutmuş olduğum docker imge ve konteynırlarını nasıl yönetebileceğim konusunda yine öncül bilgiler edinmiş bulundum.

Gelecekte docker konteynırlarının anlık işlem yüklerini alarak daha büyük ve paylaşımlı bir ağ kurabilmek, Kubernetes teknolojisi ile tanışmak konusunda büyük bir istek duymaktayım.

Özellikle devre dışı bıraktığım bazı servislere rağmen, uygulamanın tamamının işlem dışı kalmadan yoluna devam edebilmesi gibi bazı mikroservis faydalarının gözlemlemiş olmak beni memnun etti.

5- REFERANSLAR

[1]- https://www.bbc.com/turkce/haberler-dunya-45987201

[2]- https://edgy.app/machine-learning-vs-machine-reasoning-know-the-difference

[3]- https://www.portainer.io/