T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

WEB UYGULAMALARI İÇİN BULUT VE KONTEYNER TABANLI TEST OTOMASYON HİZMETİ

YÜKSEK LİSANS TEZİ

Mehmet Emin KÜÇÜKER

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ
Tez Danışmanı : Doç. Dr. Kürşat AYAN

Ağustos 2018

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

WEB UYGULAMALARI İÇİN BULUT VE KONTEYNER TABANLI TEST OTOMASYON HİZMETİ

YÜKSEK LİSANS TEZİ

Mehmet Emin KÜÇÜKER

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Bu tez 02.08.2018 tarihinde aşağıdaki jüri tarafından oybirliği / oyçokluğu ile kabul edilmiştir.

Doç. Dr. Kürşat AYAN Jüri Başkanı	Dr. Öğr. Üyesi Veysel Harun ŞAHİN Üye	Dr. Öğr. Üyesi Hayrettin EVİRGEN Üye

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim.

Mehmet Emin KÜÇÜKER

02.08.2018

TEŞEKKÜR

Yüksek lisans tez çalışmam süresince tavsiye ve eleştirileriyle bana rehberlik eden danışmanım Doç. Dr. Kürşat AYAN'a teşekkür ederim.

Tez çalışmam sırasında desteğini her zaman hissettiğim TÜBİTAK BİLGEM Test ve Değerlendirme Başkan Yardımcısı Dr. Öğr. Üyesi Ali GÖRÇİN'e ve bu çalışmayı gerçekleştirdiğim projede görev almamı sağlayan ve yardımlarını hiçbir zaman esirgemeyen TÜBİTAK BİLGEM Bilişim Teknolojileri Enstitüsü B3LAB proje yöneticisi Merve ASTEKİN'e teşekkürlerimi sunarım.

Proje çalışması sırasında tavsiyeleriyle bana yol gösteren proje danışmanım Dr. Binnur KURT'a ve proje için beraber emek sarf ettiğimiz mesai arkadaşlarıma teşekkür ederim.

Tez çalışmam süresince desteğini esirgemeyen eşim Sevde KÜÇÜKER'e ve tüm eğitim hayatım boyunca beni destekleyen aileme çok teşekkür ederim.

Ayrıca TÜBİTAK BİLGEM Bilişim Teknolojileri Enstitüsü altında yürütülen, bu çalışmanın da dahil olduğu Bulut Bilişim ve Büyük Veri Araştırma Laboratuvarı (B3LAB) projesinin desteklenmesine olanak sağlayan Türkiye Cumhuriyeti Kalkınma Bakanlığına (Proje No: 2014K121030) teşekkür ederim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

API	: Uygulama programlama arayüzü (Application Programming Interface)
CBTAaaS	: Hizmet olarak çapraz tarayıcı test otomasyonu (Cross-browser Test Automation as a Service)
CPU	: Merkezî işlem birimi (Central Processing Unit)
GUI	: Grafiksel kullanıcı arayüzü (Graphical User Interface)
HTML	: Hiper metin işaretleme dili (Hypertext Markup Language)
HTTP	: Hiper metin transfer protokolü (Hypertext Transfer Protocol)
IaaS	: Hizmet olarak altyapı (Infrastructure as a Service)
IDE	: Bütünleşik geliştirme ortamı (Integrated Development Environment)
IP	: İnternet protokolü (Internet Protocol)
JAR	: Java arşiv (Java Archive)
JAXP	: XML için Java API'si (Java API for XML Processing)
JDT	: Java geliştirme aracı (Java Development Tool)
PaaS	: Hizmet olarak platform (Platform as a Service)
POM	: Proje nesne modeli (Project Object Model)
RAM	: Rastgele erişilebilir hafıza (Random Access Memory)
REST	: Temsilî durum aktarımı (Representational State Transfer)
SaaS	: Hizmet olarak yazılım (Software as a Service)
TaaS	: Hizmet olarak yazılım testi (Testing as a Service)
URL	: Birörnek kaynak konumlayıcı (Uniform Resource Locator)
VNC	: Sanal ağ sistemi (Virtual Network Computing)
XML	: Genişletilebilir işaretleme dili (eXtensible Markup Language)

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Çapraz tarayıcı testi, test otomasyonu, bulut bilişim, dağıtık, sanallaştırma

Gelişen teknolojiler ile birlikte web uygulamalarının çalışma yükü, sunucu tarafını hafifletmek ve daha hızlı işlem gerçekleştirebilmek adına istemci tarafına kaymaktadır. İstemci tarafında kullanılabilir olan platform-tarayıcı çeşitliliği ve bunların web uygulamalarını çalıştırırken farklı davranışlar sergilemesi sebebiyle geliştirilen uygulamaların değişik ortamlarda işlevsel testlerinin gerçekleştirilmesi gerekmektedir. Bu nedenle çapraz tarayıcı uyumluluk testleri önem arz etmektedir. Bulut tabanlı yaklaşım, farklı ortamların esnek bir şekilde oluşturulması yoluyla bu testlerin uygulanması ve özellikle otomasyonu sürecinde verimliliği artırmaktadır.

Bu çalışmanın arkasındaki temel fikir, çapraz tarayıcı testinin bulut bilişimin imkanları kullanılarak açık kaynak kodlu çözümler yardımıyla otomatize bir şekilde gerçekleştirileceği özgün bir tasarım modeli sunmaktır. İşlevsel, ekran görüntüsü ve etkileşimli test olmak üzere üç farklı test tipinin sunulduğu modelde, işlevsel ve ekran görüntüsü testlerinin Selenium test aracı kullanılarak gerçekleştirilmesi tasarlanmıştır. Tasarımın uygulanması sırasında Windows ve Linux tabanlı bulut makineleri kullanılmış olup Linux makineleri üzerinde Docker teknolojisinden faydalanılarak daha esnek ve çevik bir yapı elde edilmesi planlanmıştır. Windows işletim sisteminin kısıtlarından dolayı bu test ortamları için direkt olarak OpenStack üzerinde oluşturulan bulut makineleri kullanılmıştır. Testlerin bulut makineleri ve Docker konteynerleri üzerinde dağıtık ve paralel olarak gerçekleştirilebilmesi Selenium Grid aracı ve TestNG çatısı kullanılarak sağlanmıştır.

Bu çalışmada sunulan tasarımın göstergeleri doğrultusunda, her alanda yaygınlaşan bulut bilişim teknolojisi ve Docker sanallaştırma tekniklerinin sağladığı imkanların, test otomasyonu ve tarayıcı testi alanlarına uygulanması ile bu testlerin klasik yöntemlerin aksine daha etkili ve hızlı bir biçimde gerçekleştirilmesi sağlanabilecektir.

CLOUD AND CONTAINER BASED TEST AUTOMATION SERVICE FOR WEB APPLICATIONS

SUMMARY

Keywords: Cross-browser testing, test automation, cloud computing, distributed, virtualization

With the developing technologies, the working load of web applications is shifting to the client side. Due to the variety of platforms-browsers available for the clients and different behavior of these platforms-browsers, it is important to perform the functional test in various environments for developed web applications. For this reason, cross-browser compatibility tests are crucial. The cloud-based approach improves productivity in the process of implementing and automate these tests through the flexible creation of different environments.

The idea behind this study is to present a unique model with open source solutions for cross-browser testing using cloud computing approach. In this model, three different test types were presented: Functional test, screenshot test and interactive test. Functional and screenshot tests were designed to be performed using Selenium test tool. Windows and Linux based cloud machines have been used for realization of this design model and thanks to the use of Docker on Linux machines, a much more flexible structure has been achieved. Due to the constraints of Windows operating systems, the cloud machines created on OpenStack were used directly for Windows-based test environments. Parallel and distributed tests were performed on cloud machines and Docker containers using Selenium Grid and TestNG framework.

With the findings of design model presented in this study, the opportunities provided by cloud computing technology and Docker virtualization techniques can be applied to test automation and cross-browser testing areas, and thanks to that, these tests can be performed more effectively and faster than classical methods.

BÖLÜM 1. GİRİŞ

Yazılım testi, çoğu zaman hata bulma işlemi olarak düşünülse de aslında daha çok yazılım sisteminin amaçlanan ortamda kendi tasarım şartlarını karşılayıp karşılamadığını belirlemek için yürütülen süreçler bütünüdür [1]. Yazılım dünyasının hızla gelişmesi ve değişmesinden dolayı yazılım testi bir takım açılardan hem daha zor hem de daha kolay bir hâl almaktadır [2].

Sheehan ve Young'un verdiği bilgiye göre 1990 yılında dünya nüfusunun %0.05'inden daha azı İnternet kullanırken bu oran 2010 yılında %30, 2012 yılında ise %35 seviyelerine kadar çıkmıştır [3]. İnternet kullanımındaki bu yükselişin hâliyle web uygulamalarının sayısında da benzer oranlarda artışa sebep verdiği öngörülebilmektedir. Belirtildiği gibi yazılım testi yazılımın istenilen ortamlarda çalışma durumunun gözlenmesidir. İşletim sistemi ve tarayıcıların tür ve versiyon çeşitliliği ise web uygulamaları için çalışması istenilen ortam sayısını önemli biçimde etkilemektedir. İnternet teknolojisindeki bu gelişme ve yaygınlaşma nedeniyle her ne kadar web uygulamalarının yazılım testinin daha zor bir hâle geldiğini belirtebilsek bile, yine yazılım ve bilişim dünyasındaki gelişmeler sayesinde bu zorlukların aşılmasına imkân sağlayacak çözümler üretilebilmektedir. Bulut bilişim teknolojisinin getirdiği yenilikler sayesinde klasik yazılım testi yöntemlerinin yetersiz kaldığı noktalarda kullanılmak üzere bulut tabanlı yazılım testi yaklaşımları geliştirilebilmektedir [4].

1.1. Bulut Bilişim

Bulut bilişim, yazılım ve bilişim teknolojilerinde son yılların en gözde konularından birisidir. Bulut bilişim fikri temel olarak aynı ağ içerisindeki birden fazla düğümün

aynı işi paylaşarak gerçekleştirmesi anlamına gelen dağıtık hesaplama kavramına dayanır ve bu kavrama ekonomik olarak ölçeklenebilirlik gibi yeni kazanımlar eklemiştir [5–7]. Talep odaklı kaynak kullanımı, kaynak havuzu ve bu havuzdan çoklu kullanıcı hizmeti (istemci-sunucu arasında çoktan çoğa ilişki yapısı) sağlanabilmesi, farklı platformlardan ulaşılabilirlik, çevik ve esnek yapısı ile ölçeklenebilir hizmet sunması bulut bilişimin temel özellikleridir [8–10].

Kim'e göre bulut bilişimin sağladığı temel avantajlar şu şekildedir [11];

- Yazılım, depolama ve ağ hizmeti gibi pek çok bilişim kaynağı ve bunların ekonomik giderleri üçüncül sağlayıcılara teslim edilir. Kullanıcıya yalnızca bulut ortamı ile entegre olmak kalır. Kullanıcı bu sayede veri tabanındaki kullanılabilir alan, sunucuların elektrik tüketimi ve ağdaki yoğunluk gibi endişelerden arındırılır.

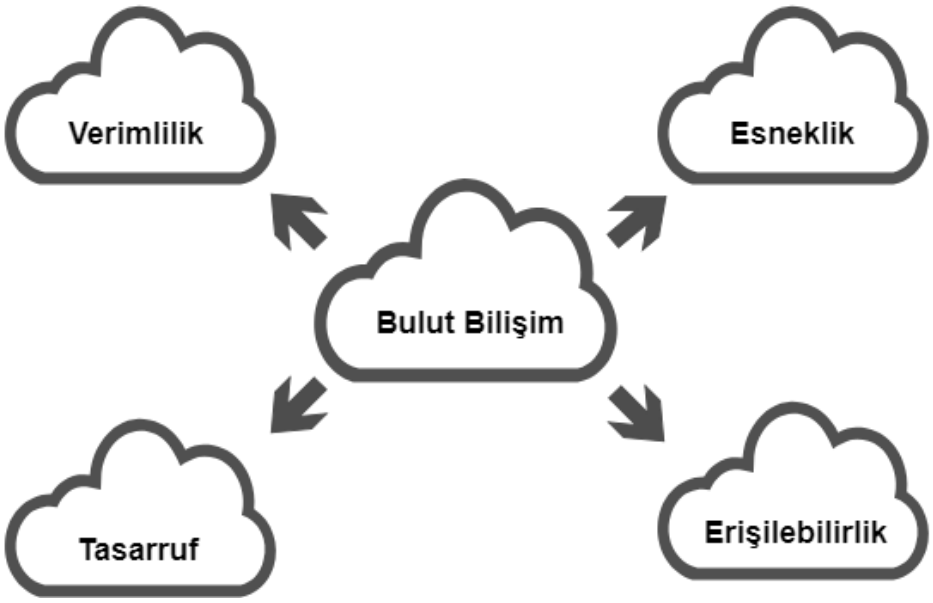
- Bilişim kaynağı kullanımı, kullanıcı talebine göre, kolayca ve esnek bir şekilde artırılabilir veya azaltılabilir.

- Kullandığın kadar öde prensibine dayandığı için kullanıcının bilişim kaynakları için ödemekle yükümlü olduğu bedel çoğunlukla geleneksel yöntemlerden çok daha düşüktür.

- Kullanıcı bulut servisine her zaman ve her yerden ulaşabilir.

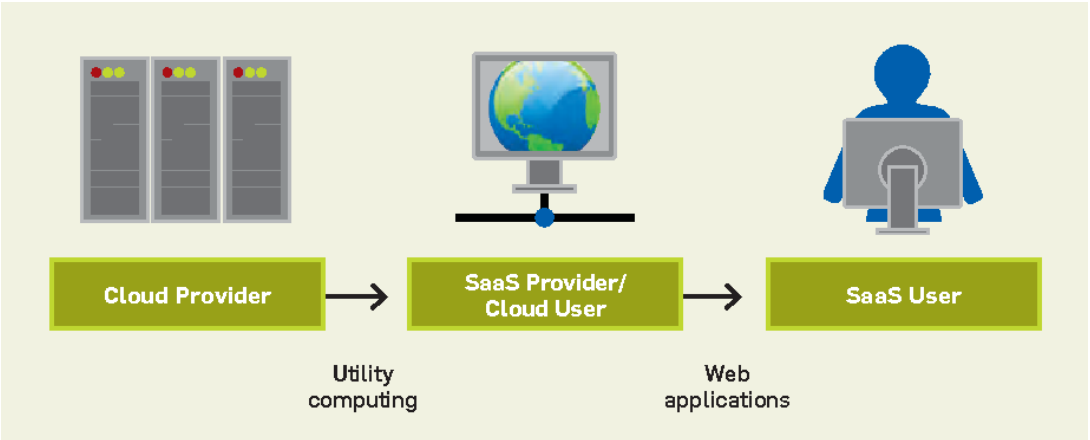
İnternet kullanımının günümüzde iyice yaygınlaşması ile birlikte sürekli ve her yerden erişim ihtiyacı çoğalmakta olup bu durumla beraber web uygulamalarının ağ yoğunlukları da artmaktadır. Dolayısıyla bulut bilişim teknolojilerinin sağladığı Şekil 1.1.'de görülen verimlilik, esneklik, tasarruf ve erişilebilirlik gibi faydalar daha önemli hâle gelmektedir. Bu doğrultuda günden güne yaygınlaşan ve gelişimini sürdüren bulut bilişim konseptinin kullanıcıya, geliştiriciye veya yöneticiye sunduğu hizmetler de çoğalmaktadır. Bu hizmetlerin her biri için "hizmet olarak" kavramı kullanılmaktadır. Her şeyin servis olarak sunulabileceği bulut ortamında bu kavramın en bilindik ve

başlıca örnekleri hizmet olarak altyapı (Infrastructure as a Service - IaaS) , hizmet olarak platform (Platform as a Service - PaaS) ve hizmet olarak yazılımdır (Software as a Service - SaaS) [10, 12].



Şekil 1.1. Bulut bilişimin temel özellikleri

Hizmet olarak yazılım: Yazılıma sahip olmadan onu kullanabilmeye imkan sağlayan servistir [13]. Şekil 1.2.'de katmanları verilen bu hizmetin sunduğu yazılım veya uygulamalara genellikle web tarayıcısı üzerinden olmak üzere bir arayüz sayesinde ulaşılır [6, 7]. Bunun en bilindik ve iyi örneklerinden biri Google tarafından sunulan Google Docs doküman uygulamasıdır.



Şekil 1.2. SaaS uygulama katmanları [5].

Hizmet olarak platform: Bulut üzerindeki makinelerin -hizmet sağlayıcısı tarafından hazır hâle getirilmiş- işletim sistemi, API (uygulama programlama arayüzü) ve geliştirme araçları gibi uygulama ve konfigürasyonlarla donatılarak platform olarak sunulmasıdır [14].

Hizmet olarak altyapı: Depolama, sunucu ve ağ gibi hizmetlerin, geleneksel barındırma işlemlerinin aksine talep odaklı ve ölçeklenebilir olmak gibi imkanlar doğrultusunda gerçekleştirilmesidir [15].

1.2. Bulut Tabanlı Yazılım Testi

Bulut bilişim teknolojilerinin bilişim dünyasına getirdiği yenilikler mühendislik alanlarında yaygın bir biçimde kullanılmaktadır. Hem yazılım hem donanım alanlarında bulut bilişimin sunduğu faydalardan yararlanmak mümkündür. Kodun, yazılımın veya uygulamanın tasarlandığı gibi çalışmasının teyidi ve uygulamanın sergileyebileceği beklenmedik davranışların önceden tespiti için gerçekleştirilen bir takım işlemler bütünü olan yazılım testi [2], bulut bilişim teknolojisinin sunduğu yeniliklerden faydalanılabilecek alanlardan birisidir.

Kaliteli bir yazılım her zaman kararlı bir şekilde çalışmalıdır ve tedarikçisine güven vermelidir. Öte yandan "hatasız yazılım yoktur". Bu nedenle hataların asgari seviyeye indirilmesi ve kullanıcıya mümkün mertebe yansıtılmaması için yazılım testi zaruri bir ihtiyaçtır. Bulut bilişim teknolojilerindeki gelişmeler ve bunların yazılım testi alanına uygulanması bu ihtiyacın daha verimli bir şekilde giderilmesine imkân tanımaktadır.

Gelişen bulut bilişim teknolojileri sayesinde yazılım testi dünyasında değişiklikler yaşanmış olup yazılım test süreçlerinin bulut bilişim teknolojilerinden faydalanarak gerçekleştirilmesi ile bulut üzerinden yazılım testi, bulut tabanlı yazılım testi veya hizmet olarak yazılım testi (Testing as a Service - TaaS) gibi kavramlar ortaya çıkmıştır [16–18].

Hizmet olarak test servislerinin esnek test ortamı sağlayabilme özelliğinden dolayı yazılım testi süreçlerine önemli ölçüde katkısı vardır [19]. Web uygulamalarının testi gibi farklı test ortamlarının kullanılmasını öngören yazılım test süreçlerinde bulut tabanlı sistemlerin kullanımı bu özelliği sebebiyle tercih edilmekte olup test sürecine önemli katkı sağlamaktadır.

1.3. Çapraz Tarayıcı Uyumluluk Testleri

Web tabanlı uygulamaların sayısının artması ve bu tarz uygulama geliştirmenin giderek daha çok tercih edilir olması web uygulamalarının test edilme ihtiyacını daha önemli hâle getirmiştir. Web uygulamalarına performans, güvenlik ve işlevsel gibi masaüstü yazılımlarda da gerçekleştirilen pek çok test tipinin uygulanması mümkündür. Bu testlerin her biri için piyasada ticari veya açık kaynaklı çeşitli araçlar bulunmaktadır. İşlevsel testlerin, yeni eklenen veya modifiye edilen her fonksiyon için tekrar edilmesi gerekmektedir. Regresyon testi [20, 21] adı verilen test tipi yeni eklenen veya üzerinde değişiklik yapılan işlevlerin sistemin çalışmasında herhangi bir olumsuzluğa neden olmadığının kanıtlanması için gerçekleştirilmektedir. Bu testlerinin tekrarlı bir yapıya sahip olmasından dolayı test otomasyonu ihtiyacı doğmaktadır. Regresyon testi için otomasyon çalışmaları gerçekleştirilebilip bu çalışmalarda kullanılacak pek çok ticari ve açık kaynaklı araç piyasada bulunmaktadır. Ancak tarayıcılar arası uyumsuzluk [22] sebebiyle, test senaryoları otomatize edilse bile, işlevsel testlerin tek ortamda koşturulması işlevlerin her kullanıcıda başarılı bir şekilde yerine getirildiğini kabul etmek için yeterli olmamaktadır. Farklı tarayıcılarda aynı sayfa işlenirken görsel bazı farklılıklar oluşabilmektedir veya bir tarayıcıda çalışan fonksiyonun bir başkasında doğru biçimde çalışmama ihtimali söz konusudur. Dolayısıyla web uygulamaları için regresyon testleri daha büyük önem arz edip, ortam değişiklikleri bu testlerde göz önünde bulundurulmalıdır. Bu doğrultuda gerçekleştirilen testler ise çapraz tarayıcı testleri veya çapraz tarayıcı uyumluluk testleri olarak isimlendirilmektedir [23].

Modern web uygulamalarının çalışma yükünü büyük oranda istemci tarafı omuzlandığı ve bunu tarayıcı üzerinde gerçekleştirdiği için çapraz tarayıcı uyumluluk

testlerinin önemi günümüzde daha çok artmıştır [23]. Üstelik istemci tarafında çalışan uygulamaların yazıldığı betik dillerinin davranışını yalnızca tarayıcı değil üzerinde çalıştığı işletim sistemlerinin tipi ve versiyonu da etkilemektedir. Bu sebeple web uygulamalarının testleri farklı işletim sistemleri üzerindeki farklı tarayıcılarda gerçekleştirilmelidir. Platform ve tarayıcı çeşitliliği, çapraz tarayıcı test yaklaşımının dayandığı temel konudur. Bu noktada bulut bilişimin sağladığı imkanlar devreye girmektedir. Çapraz tarayıcı testlerinin gerçekleştirilmesi için ihtiyaç duyulan ortam çeşitliliği bulut altyapı teknolojisinin esneklik, sürdürülebilirlik ve erişilebilirlik gibi özellikleri yardımıyla verimli ve hızlı bir şekilde elde edilip kullanılabilmektedir.

Bu çalışmada çapraz tarayıcı uyumluluk testi açık kaynak kodlu bir tarayıcı otomasyon aracı ile bulut bilişim teknolojisinin sağladığı imkanlar bir arada kullanılarak gerçekleştirilmiştir. Bulut makineleri üzerinde Docker [24] konteyner yapısı kullanılarak sanallaştırma bir adım daha ileriye taşınmıştır ve çoklu kullanıcılar veya testler için aynı makinede birbirinden yalıtılmış ortamlar sunulmuştur. Sanal ağ sistemi VNC ile sanal makinelerin grafik arabirimlerine erişim sağlanmıştır. Bu sayede kullanıcıya hem oluşturulan ortamlar üzerinde koşan testleri izleme hem de sunulan ortamlarda gerçek zamanlı test yapabilme imkânı sağlanmıştır.

BÖLÜM 2. KAYNAK ARAŞTIRMASI

Web uygulamalarında karşılaşılması muhtemel işlevsel veya görsel hataların uygulama kullanıma sunulmadan giderilmesi; kullanıcı memnuniyetini artırmak, müşteri ve itibar kaybını engellemek için oldukça önemlidir. Günümüzde son kullanıcının tercih edebileceği yüzlerce işletim sistemi ve tarayıcı kombinasyonu söz konusu olup bu kombinasyonların her birinde uygulamanın tutarsız davranışlar sergileme ihtimali vardır [25]. Web uygulamalarının çeşitli test ortamlarında uyumlu bir şekilde çalışmasını gözlemek için farklı tarayıcı test yaklaşımları bulunmaktadır. Bu tarayıcı test yaklaşımlarını yapısal olarak iki açıdan sınıflandırabiliriz. Bunlar bulut tabanlı ve geleneksel yaklaşımlardır [4]. Buna ek olarak, çapraz tarayıcı test araçları incelenirken sundukları çözümler doğrultusunda üç temel yetenek ile karşılaşılmaktadır: Ekran görüntüsü testi, işlevsel test ve etkileşimli test [26].

Çapraz tarayıcı testlerine getirilen bulut tabanlı çözümler son yıllarda artış göstermiştir. Ancak bulut bilişimin nimetlerinden henüz bu kadar yaygın bir şekilde yararlanılmadığı yıllarda da farklı tarayıcı türleri mevcuttu ve bu tarayıcılarda uyumsuzluklar yine gözlenmekteydi. Dolayısıyla bu soruna getirilmiş geleneksel çözümler bulunmaktadır. Masaüstünde aynı tarayıcının farklı versiyonlarını çalıştırabilen uygulamalar bulunduğu gibi tarayıcı testlerini otomatize edecek araçlar da mevcuttur. Bulut bilişimin bu araçlara getirdiği yeni bakış açısı ise farklı tarayıcı tür ve versiyonları ile birlikte farklı işletim sistemi kombinasyonlarının çalıştırılacağı sanal ortamların ölçeklenebilir bir şekilde kullanıma sunulmasıdır. Günümüzde her ne kadar bulut bilişim tabanlı çözümler geliştirilse de konvansiyonel çözümlere dahil edilebilecek statik analiz yöntemleri gibi akademik çalışmaların da gerçekleştirildiği görülmektedir.

2.1. Çapraz Tarayıcı Testi için Geleneksel Çözümler

Geleneksel çözümler içerisinde sınıflandırılabilecek test araçlarının en bilinenleri çevrimiçi çalışan bir masaüstü aracı olan ve Internet Explorer sürümlerinde web uygulamalarının testlerinin gerçekleştirilebileceği IETester [27] ile Firefox tarayıcısı üzerine kurulabilen Selenium IDE [28, 29] eklentisidir. IETester aracı web uygulamasının yalnızca Internet Explorer sürümlerinde çalıştırılarak manuel testlerinin gerçekleştirilmesini sağlamaktadır dolayısıyla çapraz tarayıcı testi için yeterli bir çözüm sunmaz. Selenium IDE ise kaydet ve oynat özelliği sunan bir Firefox eklentisidir ve IETester gibi bu araç da yalnızca bir tarayıcı türü için test gerçekleştirebilmektedir. Ayrıca Selenium, Firefox 55 sürümünden itibaren Selenium IDE eklentisinin çalışmayacağını duyurmuştur [28].

Statik analiz teknikleri ve model tabanlı karşılaştırmalar tarayıcı uyumsuzluklarının tespitinde kullanılabilen yöntemlerdendir. Crawljax [30, 31] model tabanlı karşılaştırma yaparak tarayıcı uyumsuzluklarını tespit etmeyi amaçlayan bir yazılımdır. Referans ve hedef tarayıcıda web sayfasını dolaşarak iki model elde edilir ve bu modeller karşılaştırılarak hedef tarayıcıdaki uyumsuzluklar tespit edilir [31]. Crawljax, fonksiyonel testten ziyade birim test mantığına daha yakın bir araçtır ve bir web sayfasına ait bütün Birörnek Kaynak Konumlayıcıların (URL) birbirileri arasındaki geçişleriyle birlikte haritasını çıkararak bir karşılaştırma gerçekleştirir. Xu ve Zeng [32] ise tarayıcı uyumsuzluklarının tespiti için statik bir yöntem öne sürmüşlerdir. Xu ve Zeng'in yöntemine göre tarayıcılar tetkik edilerek HTML5 kaynaklı uyumsuz özelliklerin veri tabanı oluşturulur ve bu veri tabanı kullanılarak web uygulamaları incelenir. Fakat bu çalışmada Xu ve Zeng Internet Explorer 11, Chrome 39 ve Firefox 35 tarayıcılarına ait yalnızca HTML5 kaynaklı uyumsuzlukların statik bir analizini gerçekleştirmiş olup işletim sistemi etkisini ise hesap etmemişlerdir [32].

Selenium WebDriver, Selenium tarafından sunulan ve Firefox eklentisinden farklı olarak diğer tarayıcılarda da fonksiyonel testleri otomatize bir şekilde gerçekleştirecek betikler yazılmasına olanak sağlayan bir API'dir [28]. Rahman [33], tez çalışmasında

Selenium WebDriver API'sinden yararlanarak tarayıcı tabanlı test otomasyonu gerçekleştirmiştir ancak farklı işletim sistemlerinin web uygulamalarının çalışmasına etkisini göz ardı etmiştir. Gmail için test otomasyonunun gerçekleştirildiği çalışmada yalnızca Windows 7 platformunda Firefox ve Chrome tarayıcıları kullanılmış olup testler dağıtık ve eş zamanlı olarak gerçekleştirilmemiştir [33].

Selenium 2.0 (WebDriver API) sürümünün yayınlanması ile birlikte Selenium Server dahili Grid işlevselliğine sahip olmuştur ve testlerin dağıtık bir şekilde koşturulmasına olanak sağlanmıştır [28]. Ayrıca, TestNG [34] sayesinde paralel test koşturma olanaklı hâle gelmiştir [35]. Dolayısıyla üzerinde farklı tarayıcı tür ve versiyonlarının çalıştığı farklı işletim sistemlerine sahip bilgisayarlardan oluşturulacak bir laboratuvar ortamında çapraz tarayıcı testlerinin pek çok kullanıcı ortamı için tek seferde gerçekleştirilmesi mümkün fakat oldukça maliyetli bir yöntemdir. Bahsedildiği gibi bir laboratuvar ortamı klasik sunucu yöntemleriyle kurulabilir ve buradan hizmet satın alınabilir. Ancak böyle bir çözüm ölçeklenebilir olmayacağı için yükün arttığı veya sunulan platformların çöktüğü durumlarda hizmetin alınamaması gibi ciddi problemlerle karşılaşma ihtimali her zaman söz konusudur ki bu şekilde hizmet sunduğu gözlenen bazı online araçlarda sonuç üretiminin oldukça yavaş gerçekleştiği gözlenmiştir. Bu şekilde tasarlanmış araçlara örnek gösterilebilecek Browsershots merkezi bir sunucu ile o sunucuya kayıtlı çok sayıda düğümden oluşan dağıtık bir tasarıma sahiptir ancak kullanıcının seçtiği ortamlardan sonuçları alabilmesi tamamen bu ortamların kurulu olduğu düğümlerin anlık durumlarına bağlıdır [36].

2.2. Bulut Tabanlı Çapraz Tarayıcı Test Araçları

Klasik sunucu yöntemlerinin yetersiz ve ilkel kaldığı noktada devreye bulut tabanlı çözümler giriyor. Klasik yöntemlerde yaşanabilecek sunucu sayısı ve yazılım kurma problemleri bulut bilişimin getirdiği ölçeklenebilirlik, kullandığın kadar öde gibi yeniliklerle büyük ölçüde azaltılmış ve bu kazanımlar test tekniklerinin de yönünü değiştirmiştir [4]. Browsershots [36] ve benzeri konvansiyonel yöntemlerle geliştirilmiş test araçlarında gözlemlenen sorunları ortadan kaldırmak için çapraz tarayıcı testini bulut tabanlı gerçekleştiren araçlar geliştirilerek piyasadaki yerlerini

almıştır. Bu araçların bulut tabanlı geliştirilmesi ile birlikte kullanıcılara sunucu endişesi taşımadan izole ortamlar sunulabilir hâle gelmiş ve etkileşimli test gibi yeni yetenekler araçlara eklenmiştir. Browserhosts [36] ve Browserbite [37] sadece URL tabanlı ekran görüntüsü testi gerçekleştirme yeteneğine sahipken, Selenium'un da destekçileri arasında bulunup alanının en popüler örnekleri olan CrossBrowserTesting [38], Sauce Labs [39] ve BrowserStack [40] gibi bulut tabanlı olarak geliştirilmiş araçlar ise ekran görüntüsü testinin yanı sıra fonksiyonel test otomasyonu ve etkileşimli test hizmetleri de sunmaktadır [26]. Fakat tümüyle açık kaynak kodlu ürünler üzerinden geliştirilmiş olmayan bu araçlar ticari ürünler olup tasarım modelleri hakkında detaylı bilgi edinilememektedir. Bu ürünlerin öne sürdüğü çapraz tarayıcı test yaklaşımında, Docker [24] teknolojisinin getirdiği yeniliklerden yararlanılmadığı için, kaynak kullanımı daha verimsiz bir biçimde gerçekleştirilmektedir.

Açık kaynak kodlu çözümler kullanılarak geliştirilen bu uygulamada bulut ortamı üzerindeki makinelerde sanallaştırmayı bir seviye daha artırmak adına Docker teknolojisinden faydalanılmıştır. Linux işletim sistemi tabanlı platformlar Docker imajları olarak düzenlenmiş ve bu ortamlar Docker konteynerleri üzerinden kullanılmıştır. Test tipi olarak ekran görüntüsü testi, işlevsel test ve etkileşimli test türlerinin tamamının sunulduğu uygulamada aynı zamanda test betiklerinin yazılıp derlenebileceği ve sistemimizde hem konfigürasyonları hem de sonuçları ile birlikte kayıt altında tutulabileceği bir tümleşik geliştirme ortamı (IDE) sunulmuş olup testlerin paralel koşturulması için kullanılması gereken TestNG konfigürasyonları otomatize edilmiştir. Bu durum geliştirilen uygulamayı, yalnızca kullanıcının bir API aracılığı ile eriştiği Selenium Grid uygulamasını hizmet olarak sunan muadillerinden farklı kılarak tam bir servis olarak yazılım (SaaS) hâline getirmektedir. Dolayısıyla son kullanıcının Selenium kurulumu veya konfigürasyonu yapmasına gerek kalmadığı gibi testlerini koşturmak adına yerel bilgisayarında kurulu herhangi bir ürüne dahi ihtiyacı olmamaktadır.

BÖLÜM 3. BULUT TABANLI ÇAPRAZ TARAYICI TEST OTOMASYONU

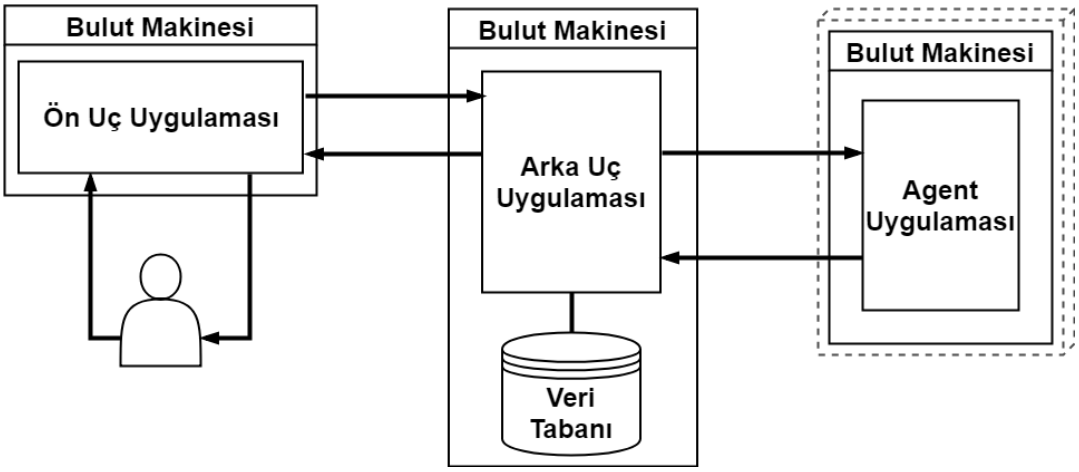
Katherine ve Alagarsamy'ye göre kullanıcı sayısının tahmin edilemediği veya kullanıcı ihtiyaçlarına göre dağıtım ortamında değişiklik olan uygulamalar için bulut test kavramı bulut bilişim ve yazılım mühendisliği alanlarında günden güne önemi artan bir konu hâline gelmiştir [4]. Dallmeier ve diğerleri, bir çalışmada sürekli artan bileşen sayısı ve bunların arasındaki entegrasyon ağının büyümesi ile daha fazla ve daha iyi test yapılmasının gerekliliğine ve dolayısıyla manuel testin her zaman pahalı olduğuna vurgu yapmışlardır [41]. Mesbah ve Prasad ise tarayıcı tür ve versiyonları ile istemci tarafı ortamlarının sayısındaki aşırı artıştan dolayı çapraz tarayıcı uyumluluğu sorununun daha da önem kazandığını belirtmiştir [23]. Bu çalışmada, bu üç konsept (bulut bilişim, test otomasyonu ve çapraz tarayıcı testi) bir araya getirilmiştir ve bulut bilişimin imkanlarından yararlanılarak çapraz tarayıcı testlerinin otomatik olarak gerçekleştirilebilmesi adına hizmet olarak test uygulaması geliştirilmiştir. Bu çalışma bulut tabanlı çapraz tarayıcı test otomasyonu (Cloud-based Cross-browser Test Automation) veya hizmet olarak çapraz tarayıcı test otomasyonu (Cross-browser Test Automation as a Service - CBTAaaS) olarak isimlendirilebilir.

Geliştirilen bu uygulamada kullanılan bütün teknolojiler, araçlar ve üçüncül ürünler için açık kaynak kodlu çözümler tercih edilmiştir.

3.1. Tasarım Modeli

Bu çalışmada önerilen bulut tabanlı çapraz tarayıcı test otomasyonu yaklaşımına ait tasarım Şekil 3.1.'de tarif edildiği gibi üç ana katmandan oluşmaktadır. Bunlar kullanıcının etkileşim hâlinde olduğu ön uç, ön uç tarafından yapılan kullanıcı

isteklerinin karşılanıp işlendiği arka uç ve test koşturma işlemlerinin gerçekleştirildiği Agent uygulamalarıdır.



Şekil 3.1. Bulut tabanlı çapraz tarayıcı test otomasyonu uygulama katmanları

3.1.1. Ön uç uygulaması

Angular ile geliştirilen ön uç tarafında proje, test grubu (Test Suite) ve test durumu (Test Case) oluşturulabilmektedir. Uygulamada test durumları için üç tip test seçeneği sunulmakta olup bunlar: İşlevsel test, ekran görüntüsü testi ve etkileşimli testtir. Test koşturma işlemi test durumları üzerinden gerçekleştirilebileceği gibi test grupları kullanılarak grubun altında tanımlanmış olan işlevsel test tipindeki test durumları için toplu olarak da gerçekleştirilebilmektedir. Koşturulacak test durumu veya test grubu için testin gerçekleştirileceği test ortamı bilgisi testi başlatmadan önce tercih edilmelidir.

3.1.1.1. İşlevsel test tipi

İşlevsel test tipi için web arayüzü üzerinden IDE görevi görecek olan bir kod editörü sağlanmakta olup bu editör üzerinden test betikleri eklenip düzenlenebilmektedir. Kaydedilen test betikleri çalıştırılabilir dosyaya çevrilerek seçilen test ortamları üzerinde koşturulur. Test sonuçlarına ait kayıtlar ve testin canlı akışı arayüz üzerinden yayımlanır ve test sonlandığında ilgili teste ait video kaydı istek doğrultusunda ön uca

iletilir. Koşturulan test sonuçlarına istenilen zamanda arayüz üzerinden erişim mümkündür.

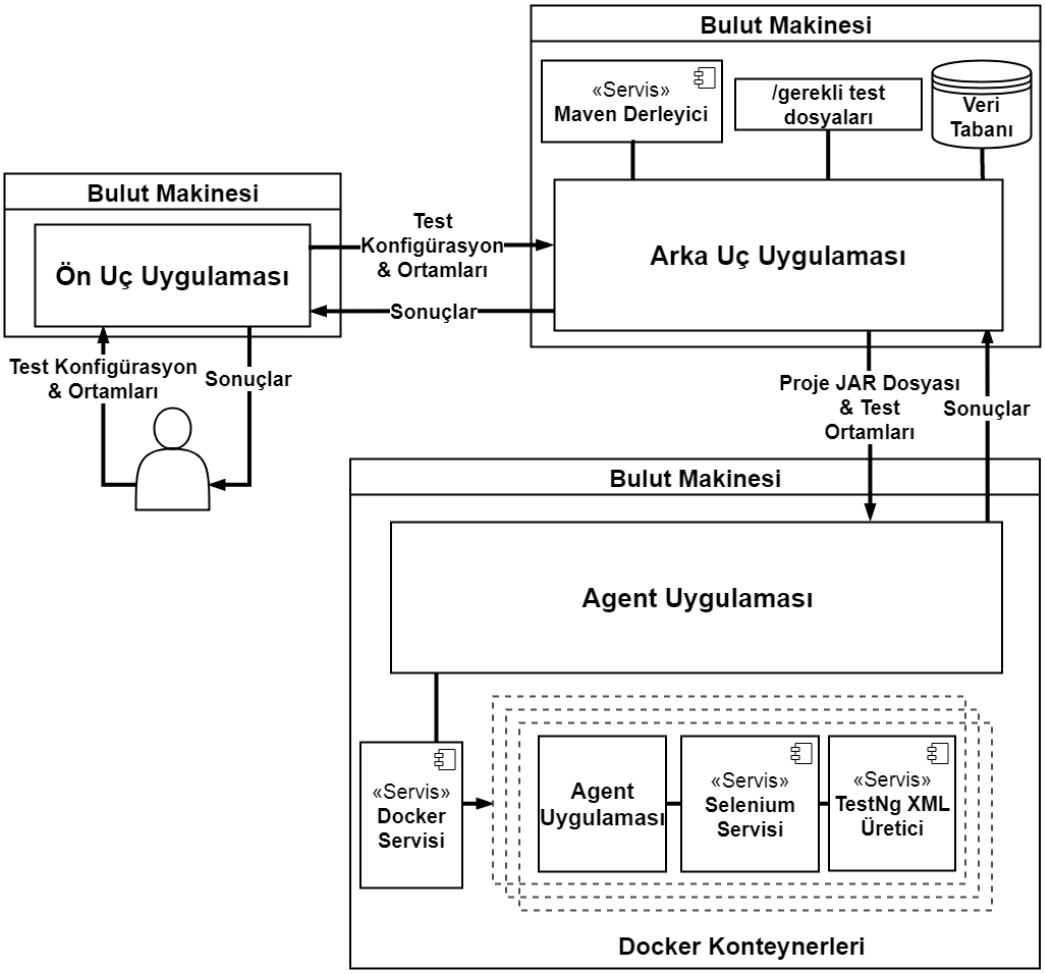
3.1.1.2. Ekran görüntüsü test tipi

Ekran görüntüsü test tipinde yalnızca ekran görüntüsü alınmak istenen URL bilgisi ve ilgili test ortamları girdi olarak beklenir. Önceden oluşturulmuş standart bir çalıştırılabilir dosya kullanılarak test otomasyon aracı yardımıyla alınan ekran görüntüleri veri tabanına kaydedilerek arayüz üzerinden sunulur.

3.1.1.3. Etkileşimli test tipi

Etkileşimli test tipinde, seçilen test ortamına arayüz üzerinden erişim sağlanmakta olup sunulan test ortamında manuel testlerin gerçekleştirilmesine imkân tanınmaktadır. VNC aracılığıyla gerçekleştirilen bu erişim için Agent tarafında açık kaynak kodlu bir çözüm olan noVNC uygulaması kullanılmış olup bu sayede kullanıcı tarafında web tarayıcılarının VNC istemcisi olarak çalışmasına olanak sağlanmıştır [42]. Test ortamı hazır hâle geldikten sonra bu ortama ait IP ve port bilgilerinden oluşan URL bilgisi ön uca iletilir. Test ortamına arayüz üzerinden erişim, ön uca iletilen bu URL bağlantısı kullanılarak sağlanmaktadır.

Uygulama mimarisi Şekil 3.2.'de tarif edilen bulut tabanlı çapraz tarayıcı test otomasyonu yaklaşımında, test oluşturma ve derleme ile ilgili süreçlerin gerçekleştirilmesi için arka uçta Maven [43] tabanlı bir derleyici servisi geliştirilmiştir. Agent ucunda ise sanallaştırmayı artırarak testlerin koşturulması sırasında daha verimli bir çözüm sunma hedefi ile Docker konteyner yapısından yararlanılmış olup bu yapının kontrolünü sağlayan Docker servisi yazılmıştır. Docker konteynerleri içerisinde testlerin koşturulmasında kullanılan Selenium ve TestNG servisleri geliştirilmiştir.



Şekil 3.2. Bulut tabanlı çapraz tarayıcı test otomasyonu uygulama mimarisi

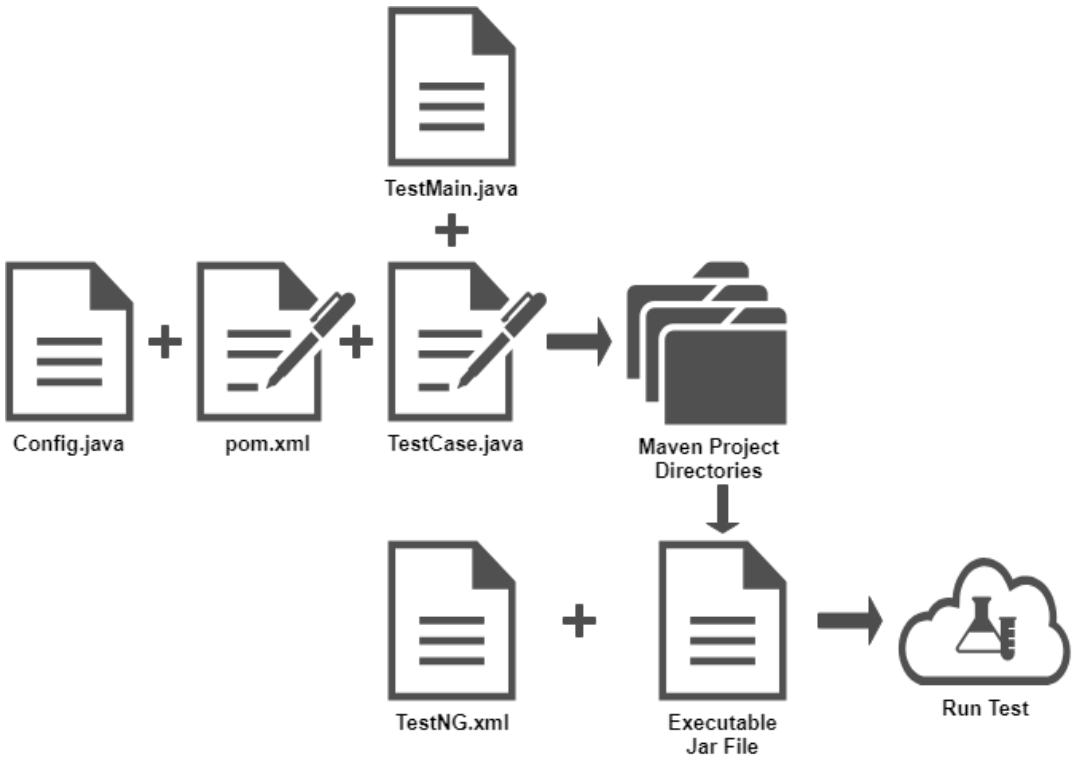
3.1.2. Arka uç uygulaması

Proje, test grubu veya test durumu için arayüz aracılığıyla gerçekleştirilen görüntüleme, oluşturma, silme veya güncelleme istekleri ile test ortamı seçimi ve test betiği operasyonları ön uçtan arka uca aktarılır. HTTP istekleri (GET, POST, PUT, DELETE) kullanılarak gerçekleştirilen bu aktarım sonucunda, REST API [44] uygulama geliştirme çatısı olan Spring [45] ile geliştirilen arka uçta ilgili süreçler işletilir.

3.1.2.1. Maven ile çalıştırılabilir test dosyası oluşturma

Maven, Java dilinde geliştirilen yazılım projelerinin derleme ve yönetiminde kullanılan bir Java Geliştirme Aracıdır (JDT) [43]. Maven projelerinde, gerekli

eklentiler ve kütüphaneler içeri aktarılmak yerine projeye ait bir XML (pom.xml) dosyasında bağımlılıklar olarak tutulmaktadır. Bu eklenti ve kütüphaneler internet aracılığıyla havuzdan çekilip projeye otomatik olarak enjekte edilir. Bu uygulama Maven uyumlu Spring çatısı kullanılarak geliştirilmiştir. Aynı zamanda Şekil 3.2.'de görüldüğü gibi betik derleme (Maven Derleyici) modülünde ve Şekil 3.3.'te tarif edilen çalıştırılabilir test dosyasının oluşturulması sürecinde de Maven aracından faydalanılmıştır.



Şekil 3.3. Maven ile çalıştırılabilir JAR dosyası oluşturma akışı

İşlevsel test tipinde, kullanıcıdan alınan girdiler doğrultusunda Şekil 3.3.'te gösterilen akış ile arka uçta teste ait bir Maven projesi oluşturulmaktadır. Arayüz aracılığı ile test betiği eklenip betik derleme modülü üzerinden Maven komutu kullanılarak derlenebildiği gibi test betiğinde kullanmak üzere projeye kütüphaneler de dahil edilebilmektedir. Kod editörü üzerinden sunulan Proje Nesne Modeli (POM) dosyasında değişiklikler yaparak test betiğinde kullanılmak istenen metotların ait olduğu kütüphaneler betiğe eklenebilmektedir. Aynı zamanda her projede sabit olarak bulunan konfigürasyon sınıfı, Şekil 3.2.'de görülen arka uç kodunun çalıştığı bulut

makinesindeki ilgili dizinden kopyalanarak, test için oluşturulan Maven projesi dizinine eklenmektedir. Test betiği bu konfigürasyonlar dikkate alınarak yazılmalıdır. Konfigürasyon sınıfı kullanıcının tercih ettiği tanımlı ortamlardan oluşturulan TestNG XML dosyasındaki parametrelerin Selenium test betiğine aktarılmasını sağlamaktadır. Bu parametreler, testler dağıtık olarak koşturulurken, tercih edilen tanımlı test ortamlarına göre oluşturulan düğümlerin özellikleri ile eşleşmekte ve böylece test koşturma işlemi otomatik olarak gerçekleştirilmektedir.

Yeni eklenen veya üzerinde değişiklikler yapılan betik dosyaları, Maven yapısına uygun olarak arka uçta teste özel oluşturulan dizinde bulunması gereken klasörlerin içerisinde oluşturulur. Gerekli dosyalar arasında tutulan ve çalıştırılabilir JAR dosyası için başlangıç görevi görecek olan Şekil 3.4.'te gösterilen ana (main) Java sınıfı da proje dizinine eklenir. Oluşturulan proje, Java kodu üzerinde meydana getirilen proses aracılığıyla, aşağıda gösterilen Maven komutu kullanılarak JAR dosyası hâline getirilir:

mvn clean compile assembly:single

Arka uçta oluşturulan JAR dosyası, tercih edilen test ortamlarına ait bilgilerle beraber bu ortamlardan herhangi birinde (dağıtıcı görevini de bu makine üstlenecektir) çalışan Agent koduna gönderilir.

```
TestListenerAdapter tla = new TestListenerAdapter();
TestNG testng = new TestNG();
testng.setTestClasses(new Class[] { configClass , scriptClass });
testng.setUseDefaultListeners(false);
testng.addListener(tla);
List<String> suites = Lists.newArrayList();
suites.add("./testng.xml");
testng.setTestSuites(suites);
testng.run();
```

Şekil 3.4. Ana (main) Java sınıfı

Ekran görüntüsü test tipinde kullanıcıdan test ortamları ile birlikte yalnızca anlık ekran alıntısını görmek istediği sayfaya ait URL bilgisi girdi olarak beklenmektedir. Dolayısıyla aynı kullanıcının oluşturacağı farklı testler veya farklı kullanıcıların oluşturacağı testlerde test projesi için sadece URL adresi değişmektedir. Bu nedenle URL adresinin ana metot aracılığıyla dışarıdan Şekil 3.5.'te gösterildiği gibi DataProvider [34] parametresi olarak alınması sağlanıp standart bir JAR dosyası oluşturularak her defasında JAR dosyasını yeniden oluşturmanın getirdiği zaman maliyetinin önüne geçilmiştir.

```
In main Java file:
@DataProvider (name = "dataProvider1")
public Object[ ][ ] createData1 () {
   return new Object[ ][ ] {
         { url }
   };
}
In test script file:
@Test (dataProvider = "dataProvider1")
```

Şekil 3.5. Ekran görüntüsü test tipi DataProvider kod parçası

3.1.3. Agent uygulaması

Agent uygulamasının servis olarak çalışır vaziyette bulunacağı bulut makinelerinde sanallaştırmayı bir adım ileriye taşımak adına Docker teknolojisinden faydalanılarak test ortamlarına ait konfigürasyonlar Docker imajları hâline getirilip bu makineler üzerine kaydedilmiştir.

İşlevsel test, ekran görüntüsü testi ve etkileşimli test tiplerinin tamamı için hizmet sağlamayı hedefleyen uygulamada işlevsel test ve ekran görüntüsü testi Selenium aracı kullanılarak gerçekleştirilmiştir. Selenium aracının uygulamada merkezi bir önemi olduğu için geliştirmeye başlamadan önce uygulamanın ihtiyaçları doğrultusunda

Selenium aracına dair konsept kanıtlama çalışması gerçekleştirilmiş olup Selenium Grid ile TestNG çatısı incelenmiştir.

3.1.3.1. Test ortamları ve Docker

Bulut tabanlı test otomasyonu uygulamalarında tercih edilen her test ortamı, Şekil 3.1.'de gösterilen yapı doğrultusunda, bulut üzerinde çalıştırılan bir sanal makineye karşılık gelmektedir. Ancak bu çalışmada test ortamlarının daha esnek yapıya kavuşabilmesi adına, Şekil 3.2.'de görüldüğü gibi, Docker teknolojisinin kullanımı ile sanallaştırmanın bir seviye daha ileriye taşınması önerilmiştir.

Tarayıcı tür ve versiyonları için ayrı Docker imajları üretilerek bunlar bulut makinesi oluşturulurken temel alınacak sanal makine üzerinde saklanmıştır. Dolayısıyla bulut üzerinde oluşturulacak her yeni makinede bu imajlar konteyner üretmek için hazır olarak bekleyecektir.

Tanımlı test ortamları bulut makinesi üzerinde hazırlanmış Docker imajlarına karşılık gelmekte olup bu imajları barındıran bulut makinesinin anlık bellek kopyası bulut altyapısı üzerinde üretilmiştir. Bu kopya kullanılarak aynı konfigürasyonda sanal makineler oluşturulmaktadır. Oluşturulan sanal makinelere ait IP adresleri veri tabanında tutulan sanal makine havuzunda kayıt altına alınmaktadır. Testler, bu havuzdan çekilen IP adresleri ile ulaşılan bulut makineleri üzerinde koşmaktadır.

Arka uç - ön uç, arka uç - Agent, Agent – Agent uygulamaları arasındaki haberleşme, IP ve port bilgileri sayesinde REST API aracılığı ile gerçekleştirilmektedir. Arka uçta oluşturulan test Agent'a gönderilirken havuzdan çekilen IP'lerden bir tanesi üzerinde dağıtıcı (hub) görevinin gerçekleştirileceği bulut makinesine ait olacaktır ve arka uç bu makineye istek gönderecektir.

İzole test ortamları oluşturmak adına, ortamların dosya sistemi ve masaüstü arayüzleri Docker teknolojisi kullanılarak birbirinden ayrılmıştır. Tarayıcılarının çeşitli sürümlerine göre hazırlanan -Docker konteynerlerinin başlatılacağı- Docker imajları

Şekil 3.6.'daki temel Docker dosyası (Dockerfile) üzerinde düzenlemeler yapılarak derlenmesi ile üretilmektedir.

```
FROM ubuntu:14.04
RUN apt-get -y install \        # All dependencies like vnc4server
# install noVNC
COPY noVNC.zip /opt/
RUN cd /opt &&  unzip noVNC.zip
# install java
RUN apt-get -y install openjdk-8-jre
# copy environments
COPY    agent.jar /root
COPY    run-services.sh /root
COPY    geckodriver /root     # or chromedriver for chrome
COPY    selenium-server-standalone-3.4.0.jar /root
COPY # firefox or chrome deb file to path
RUN dpkg -i # for copied firefox or chrome deb file
ENTRYPOINT  ["bash","/root/run-services.sh"]
# set port and resolution parameters for run-service.sh script
CMD ["6080","8001", "1920x1080"]
```

Şekil 3.6. Docker imajları için temel Dockerfile dosyası

Başlatılacak Docker konteynerlerine arayüz (tarayıcı) üzerinden görsel olarak erişim sağlanabilmesi sanal ağ sistemi (VNC) ile gerçekleştirilmiştir. Bunun için açık kaynak kodlu üçüncül bir çözüm olan noVNC [42] yazılımı tercih edilmiştir.

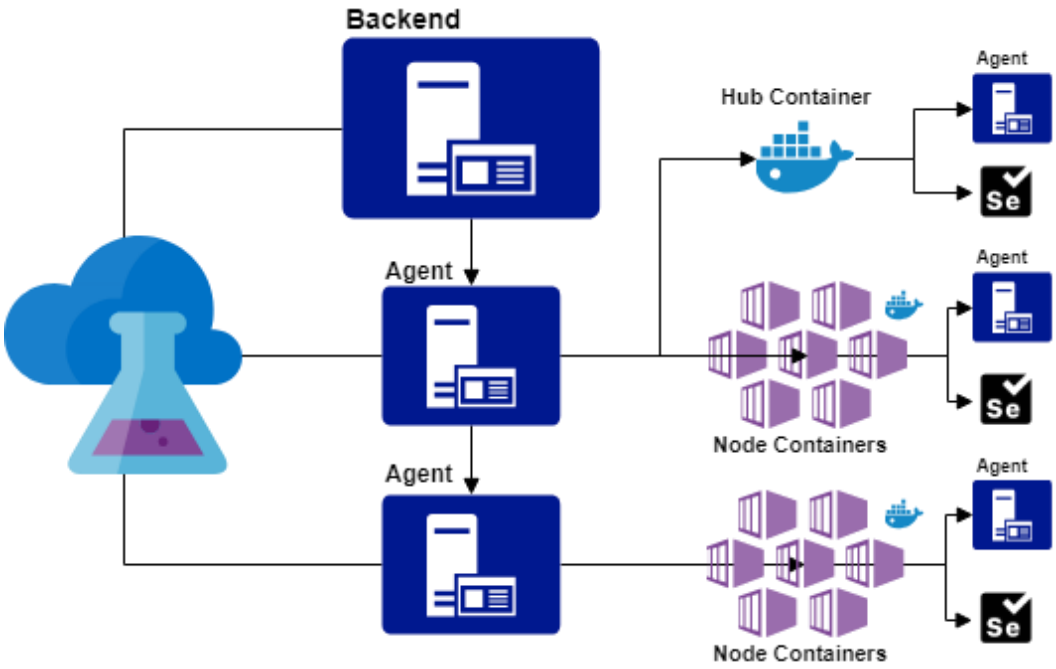
Docker konteyner içerisinde çalışacak olan Agent ve noVNC uygulamalarının komutları Şekil 3.7.'de gösterildiği gibi bir betik içerisine yazılmıştır. Bu betik Şekil 3.6.'da gösterilen Dockerfile içerisinde ENTRYPOINT komutu ile çalıştırılır. Bunun sebebi konteynerin bu betik ile birlikte başlatılmasının istenmesidir. Dockerfile hazırlanırken noVNC uygulamasını çalıştıran betik içerisindeki komut satırında çözünürlük parametreleri kullanıcı tarafından girdi olarak alınacak şekilde

tasarlanmıştır. Bu sayede aynı Docker imajından istenilen her çözünürlükte Docker konteyner, imaj yeniden derlenmeden çalıştırılabilmektedir.

```
/usr/bin/vncserver
/usr/bin/vncserver -kill :1
rm /tmp/.X11-unix/X*
rm /tmp/.X*-lock
/usr/bin/vncserver :1 -geometry $3 -depth 24  &
cp /opt/noVNC/vnc_lite.html /opt/noVNC/index.html
/opt/noVNC/utils/launch.sh --listen $1 --vnc localhost:5901  &
java -Dserver.port=$2 -Dserver.address=0.0.0.0 -jar /root/agent.jar
export DISPLAY=:1
```

Şekil 3.7. Agent ve noVNC uygulamalarının çalıştırıldığı run-service.sh betiği

Arka uçtan Agent'a HTTP isteği gönderilirken aynı zamanda sanal makine havuzundan çekilen, testin üzerinde koşacağı, diğer makinelere ait IP bilgileri de aktarılmaktadır. İsteği alan Agent kodu öncelikle Şekil 3.8.'de görüldüğü gibi kendi üzerinde -dağıtıcı görevi üstlenecek olan- Docker konteynerini başlatır. Arka uç tarafından kendisi üzerinde oluşturulması talep edilen test ortamları var ise onlara ait Docker konteynerlerini de başlatır. Agent kodu bu işlemleri gerçekleştirdikten sonra kendisine ulaşan talep içerisinde başka makinelere ait IP bilgileri var ise o makinelere düğüm oluşturma isteği gönderir.



Şekil 3.8. Docker konteyner oluşturma akışı

VNC sunucusu, Agent ve Selenium uygulamalarının her biri kendilerine atanmış portları dinleyerek istekleri karşılar. Bir bilgisayar üzerinde oluşturulacak Docker konteynerlerine yerel olarak erişim sağlanabilir. Konteynerlere başka bir bilgisayardan ulaşmak için üzerinde bulunduğu, dış dünya ile bağı olan, bilgisayarın IP adresi kullanılır. Konteynerin içinde kullanılan uygulamalara istenilen portlar atanabilir ancak üzerinde bulunduğu makineden başka makinelerin de erişimini sağlamak için atanmış her port konteynerin üzerinde bulunduğu makinede belirlenecek bir porta yönlendirilmelidir. Bu sayede yönlendirilen bu port kullanılarak Docker konteynerinin içerisindeki uygulamaya farklı bilgisayarlardan da erişilebilir. Bu yaklaşımda, test ortamları ile haberleşme ihtiyacı olduğundan, Docker konteynerler çalıştırılırken konteyner içerisinde kullanılacak Agent, Selenium ve VNC uygulamaları için port yönlendirmeleri aşağıdaki komutta gösterilen biçimde belirtilmiştir:

docker run -i -p 4001:4001 -p 4002:4002 -p 4003:4003

Bunun bir sonucu olarak Agent tarafında, arka uçta sanal makine IP'leri için kullanıldığı gibi bir havuz sistemi ihtiyacı portların kontrolü açısından doğmuş olup

Agent kodunda veri tabanı kullanılmadığı için bu noktada bellek üzerinde bir veri tabanı uygulaması tercih edilmiştir.

Agent kodu yalnızca Docker konteynerlerini ayağa kaldırmakla görevli olmayıp aynı zamanda -işlevsel ve ekran görüntüsü test tipleri için- konteynerler üzerinde çalışacak ve testlerin dağıtık biçimde koşturulmasına olanak sağlayacak olan Selenium Server Standalone uygulamasının kontrolünü de gerçekleştirir. Dolayısıyla konteynerlerin içerisinde de Agent uygulamasına yer verilmiştir. Arka uç tarafından gelen ilk talebin karşılandığı sanal makinede Şekil 3.8.'de gösterildiği gibi dağıtıcı görevi üstlenecek bir Docker konteyneri başlatılır. Ardından test ortamlarına karşılık gelecek ve düğüm görevi üstlenecek olan Docker konteynerlerin veya bulut makinelerinin hazırlanması için işlemler ve istekler gerçekleştirilir.

Bütün düğümler ve dağıtıcının hazır duruma gelmesi üzerlerinde çalışan VNC ve Agent uygulamalarının "başlatıldı" durumuna geçmesi ile gerçekleşir. Düğümler ve dağıtıcı arasındaki haberleşmenin başlatılabilmesi için bu kontrol mekanizması önem taşımaktadır. Bu mekanizma Java WatchService API'si kullanılarak Şekil 3.9.'da gösterildiği şekilde tasarlanmıştır.

```
function
path.register for ENTRY_CREATE and ENTRY_MODIFY
for infinitely
        watchKey poll for seconds
        if key is not null
                for every poll events
                        watchEvent and resolve files
                        for every container on instance
                                for every line in log file
                                        if line contains start information of Agent and VNC
                                        then break
        if other instances are also ready
        then break
End the function.
```

Şekil 3.9. Docker Agent ve VNC uygulaması için watcher algoritması

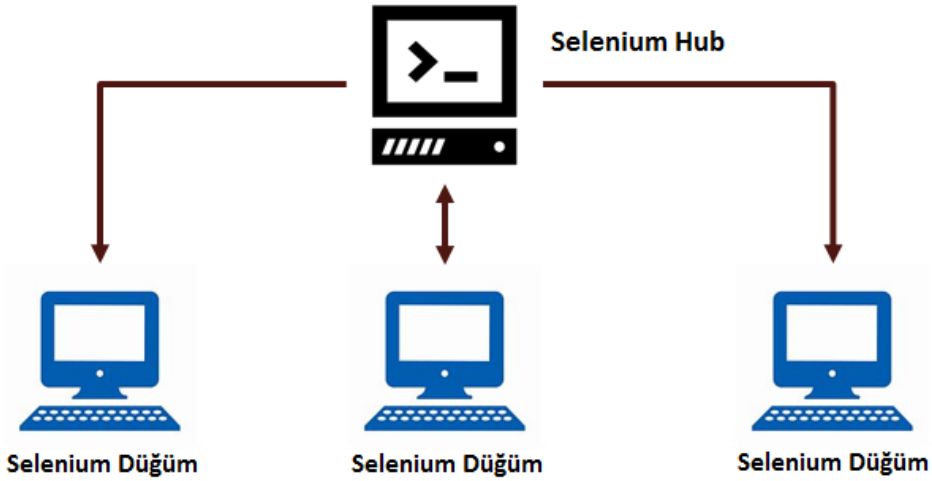
Test ortamlarının tamamı hazır duruma geldiğinde dağıtıcı görevi üstlenecek ortamda çalışan Agent uygulamasına Selenium Server Standalone uygulamasını dağıtıcı görevi ile başlatması üzere talepte bulunulur ve kendisine bağlı olacak düğüm bilgileri talep ile birlikte iletilir. Bu konteynerde Selenium Server Standalone (Selenium Grid) uygulaması dağıtıcı rolü ile başlatılır, dağıtıcıya bağlı düğümlere bu uygulamayı düğüm (node) rolü ile başlatmalarına dair talepte bulunulur ve bağlantı durumları yine Şekil 3.9.'daki gibi WatchService API'si ile takip edilir.

Dağıtıcı ve düğümler hazır duruma geldiğinde ise yine dağıtıcı üzerinden arka uç tarafından teslim edilen teste ait JAR dosyası dağıtıcı üzerinde çalıştırılır.

3.1.3.2. Selenium Grid ve TestNG çatısı

Çapraz tarayıcı testlerini gerekli kılan durum tarayıcı ve işletim sistemi çeşitliliği ile bu çeşitlilikten doğan uyumsuzlukların artmasıdır. Dolayısıyla çapraz tarayıcı testlerinin bir numaralı gereksinimi ortamlardır.

Çok sayıda bilgisayar ve bunlar üzerinde çalışan tarayıcılardan oluşan bir ortamda testlerin gerçekleştirilmesi dağıtık hesaplama kavramına dayanır. Selenium Grid web uygulamaları için bu görevi üstlenen bir araçtır ve Şekil 3.10.'da görüldüğü gibi test betiklerinin dağıtık olarak birden fazla ortamda koşturulabilmesine olanak sağlar. TestNG ise notasyonları desteklemesi, test durumlarının gruplanmasına olanak sağlaması, esnek konfigürasyon, veri güdümlü test imkanı, ve en önemlisi paralel test koşturmayı mümkün kılması ile tercih sebebi olan bir test otomasyon çatısıdır [35, 46].



Şekil 3.10. Selenium Grid mimarisi

Testlerin paralel olarak koşturulmasına olanak sağlayan TestNG çatısı aynı zamanda tek bir XML dosyası sayesinde konfigürasyonları veri güdümlü olarak teste enjekte edebilmeye imkan verir [34]. Bu çalışmada XML dosyası kullanıcının tercih ettiği test ortamlarından otomatik olarak üretilmiştir. XML dosyasını otomatik olarak üretmek için XML işlemlerinde kullanılan bir Java API'si olan JAXP kullanılmıştır. Kullanıcının tanımlayacağı test durumu için seçtiği her ortam XML dosyasındaki bir test etiketine karşılık gelecek şekilde "<test suite>" etiketinin altında Şekil 3.11.'de olduğu gibi sıralanmaktadır. "<test>" etiketi altında ise karşılık geldiği test ortamının bilgileri parametre olarak belirtilmektedir.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
- <suite thread-count="2" parallel="tests" verbose="1" name="Test Suite">
  - <test name="Linux / Chrome - 64">
        <parameter name="hubPort" value="4001"/>
        <parameter name="deviceType" value="Desktop"/>
        <parameter name="operatingSystemType" value="Linux"/>
        <parameter name="browserType" value="Chrome"/>
        <parameter name="browserVersion" value="64"/>
        <parameter name="resolution" value="1920x1080"/>
      - <classes>
            <class name="test.java.TestNG.Config"/>
            <class name="test.java.TestNG.TestCase1"/>
        </classes>
    </test>
  - <test name="Linux / Firefox - 59">
        <parameter name="hubPort" value="4001"/>
        <parameter name="deviceType" value="Desktop"/>
        <parameter name="operatingSystemType" value="Linux"/>
        <parameter name="browserType" value="Firefox"/>
        <parameter name="browserVersion" value="59"/>
        <parameter name="resolution" value="1080x720"/>
      - <classes>
            <class name="test.java.TestNG.Config"/>
            <class name="test.java.TestNG.TestCase1"/>
        </classes>
    </test>
</suite>
```

Şekil 3.11. TestNG XML dosyası

Selenium Grid, dağıtıcı ve düğümler üzerinde çalışan Selenium Server Standalone uygulamasının bağlantısı sayesinde testleri dağıtık biçimde yürütür. Dağıtıcı ve düğümleri birbirine bağlayabilmek için, düğüm üzerinde Selenium Server Standalone uygulamasını çalıştırırken, dağıtıcı makinesine ait IP ve port bilgileri parametre olarak belirtilmelidir. Dağıtıcı ve düğümlerdeki Agent kodları aşağıdaki komutlarda gösterildiği gibi ilgili parametrelerle birlikte Selenium Server Standalone uygulamalarını başlatır:

```
java -Duser.language=en -Duser.country=US
-jar selenium-server-standalone-3.4.0.jar
-role hub -port 4001 -host X.X.X.X
```

```
java -Duser.language=en -Duser.country=US
-Dwebdriver.firefox.bin=/usr/bin/firefox
-Dwebdriver.gecko.driver=/home/user/geckodriver
```

```
-jar selenium-server-standalone-3.4.0.jar
-role node -port 4002 -hub http://X.X.X.X:4001/grid/register
-browser browserName=firefox,version=59-1080x720-Linux
```

Düğümlerin her biri bir test ortamına karşılık gelir ve TestNG XML dosyasının "<test>" etiketiyle eşleşir. Selenium betikleri dağıtıcı tarafından başlatılır ve test koşmadan önce düğümlerin işletim sistemi ve tarayıcı türü gibi özellikleri ile otomatik olarak eşleşir. Ancak, Selenium test otomasyon aracı tarayıcı sürümünü ve ortam çözünürlüğünü otomatik olarak eşleştiremez. Bu nedenle, test ortamlarıyla düzgün bir eşleşme gerçekleştirebilmek amacıyla tarayıcı sürümleri ve ortam çözünürlükleri, yukarıdaki komutta gösterildiği gibi, düğüm üzerindeki Selenium Server Standalone uygulamasına parametre olarak eklenmiştir.

Dağıtıcı ve düğümler üzerinde, bu görevlerle başlatılan Selenium Server Standalone uygulamaları hazır duruma geldikten sonra test koşturma işlemi gerçekleştirilerek işlevsel test tipi için düğümler üzerinde Selenium tarafından üretilen test sonuç kayıtları arka uca ve ardından ön uca iletilir. Ekran görüntüsü test tipinde ise açık kaynak kodlu bir Selenium eklentisi olan Selenium Shutterbug [47] kullanılarak üretilen ekrana ait anlık alıntılar dağıtıcı tarafına kaydedilmektedir. Hangi ekran alıntısının hangi ortama ait olduğu bilgisi JAR dosyası oluşturulurken kullanılan betik içerisinde, ekran alıntı isminin TestNG XML dosyasından edinilen test ortamı parametreleri kullanılarak oluşturulması sayesinde ayırt edilmektedir. Ekran alıntısının ismi ve kaydedileceği yol aşağıdaki komutta gösterildiği gibi Selenium Shutterbug eklentisinin shootPage fonksiyonuna parametre olarak alınmaktadır:

```
Shutterbug.shootPage(driver, ScrollStrategy.BOTH_DIRECTIONS, 500, true)
.withName(environmentName).save("/tmp")
```

3.2. Uygulama Detayları

Bir önceki bölümde tasarım modeli tarif edilen bulut tabanlı çapraz tarayıcı test otomasyon yaklaşımının uygulanması sırasında, bulut altyapı servisi olarak muadil

uygulamaların aksine açık kaynak kodlu bir proje olan OpenStack [48] tercih edilmiştir. Tasarımın uygulanması sırasında, Docker ve Selenium servisleri geliştirilirken, bazı sorunlarla karşılaşılmıştır. Karşılaşılan sorunlar, bu sorunlara üretilen çözümler ve bulut altyapısı tarafında gerçekleştirilen çalışmalar bu bölümde açıklanmıştır.

3.2.1. OpenStack bulut altyapısı

Bulut altyapısı olarak tercih edilen OpenStack üzerinde bulut makinesi oluşturma ve silme gibi işlemler OpenStack'in hesaplama bileşeni olan Nova sayesinde gerçekleştirilmiştir [49]. Bu işlemler, Python betikleri kullanılarak Nova istemcisi komutları sayesinde yürütülmektedir. Keystone, kimlik doğrulamakta kullanılan bir OpenStack servisidir [49, 50]. OpenStack üzerinde çalışan Keystone servisi aracılığıyla kimlik doğrulama sağlanarak OpenStack hesabına erişilmektedir. Hesaba erişim sağlandıktan sonra, Nova projesi tarafından sağlanan bir RESTful HTTP servisi olan OpenStack Compute API'nin komutları kullanılarak altyapı tarafına bulut makinesi oluşturup silme işlemleri için isteklerde bulunulmakta ve işlemler gerçekleştirilmektedir [48].

Bulut üzerinde makine oluşturma ve silme işlemlerinin gerçekleştirilmesinde Şekil 3.12.'de gösterilen metot kullanılmıştır. Bu algoritma veri tabanında bulundurulan kullanılabilir bulut makinesi sayısını sabit tutmaya yönelik geliştirilmiştir. Başlatılan her test için tercih edilen ortamların işletim sistemi bilgisine göre bulut ortamında kullanılan makine kadar yeni makine testin yürütülmesi sırasında oluşturulmaktadır. Bu yaklaşım ile birlikte bir test koşturulurken gelebilecek başka test istekleri için, hâlihazırda müsait Agent makinesi kalmaması durumunda, oluşabilecek bekleme süresini kısaltmak ve isteklerin birikmesinin önüne geçmek hedeflenmiştir. Fakat çok yoğun kullanım senaryosu için yine de kuyruk sisteminin uygulanması gerekmektedir. Dolayısıyla gelen istekler önce kuyrukta toplanmaktadır. Kullanılabilir durumdaki bulut makineleri ile başlatılan teste ait ortam bilgisinin karşılaştırılması sonucunda kuyruktan sıra ile beklemeye alınan test istekleri çekilip test başlatılmaktadır. Ayrıca

zaman tabanlı bir mekanizma işletilerek uzun süredir kullanılmayan bulut makinelerinin silinip bulut ortamında gereksiz yer işgal etmelerinin önüne geçilmiştir.

```
Function
find testCase by id through testCaseRepository
if testCaseStatus is not Completed
    insert testCase to the Queue
foreach element in Queue
    if available cloud instances in database are adequate for testCase
        allocate instances from cloudInstancePool
        create instances on cloud as much as allocated from cloudInstancePool
        initiate the related testCase
    else break
End the function.
```

Şekil 3.12. Bulut makinesi oluşturma algoritması

3.2.2. Docker ve Windows test ortamı

Bu çalışmada Linux test ortamları Docker kullanılarak oluşturulmuştur. Docker sayesinde bu ortamlar için oluşturulan bulut makineleri üzerinde birden fazla test ortamı oluşturma imkânı doğmaktadır.

İlk aşamada Linux ve Windows ortamlarını sunmayı hedefleyen uygulamamızda Windows ortamları için Docker kullanımı bu işletim sisteminin kısıtlarından dolayı gerçekleştirilememiştir. Farklı işletim sistemi tür ve sürümlerine olan ihtiyaç temel olarak Selenium Webdriver'ın çalışma durumundan kaynaklanmaktadır. Ancak Selenium aracında Linux tabanlı işletim sistemleri için ayrı platform tanımlaması yapılmamış olup bu sistemlerde aracın istikrarlı bir şekilde çalıştığı öne sürülmektedir. Dolayısıyla Linux ortamları için farklı işletim sistemi sürümlerine ihtiyaç duyulmazken Windows için aynı durum geçerli değildir. Bu durum Selenium Webdriver'ın çalışmasını etkileyen Windows sürümleri için çapraz tarayıcı testinin ayrı ayrı gerçekleştirilmesi ihtiyacını doğurmaktadır. Fakat Windows üzerinde, Linux

işletim sisteminin aksine, Docker imajları yalnızca nanoserver veya windowscoreserver temel imajları üzerinden oluşturulabilmektedir [51]. Bu çalışmada, Docker kullanılarak Windows'a ait faklı sürümlerden test platformları oluşturulamaması kısıtından dolayı Windows ortamları için kullanıcıya direkt bulut makineleri sunulmuştur. Tasarımın konsept kanıtlama çalışmaları Windows Server 2012 R2 kullanılarak gerçekleştirilmiştir.

3.2.3. Selenium konsept çalışması ve Video Node uygulaması

Bu çalışma, Selenium 3.4.0 sürümü baz alınarak tasarlanmış olup Chrome için 56 ve 67 arası sürümler, Firefox için ise 54 ve 60 arası sürümler sunularak bu sürümlerin uyumlu çalışacağı chromedriver ve geckodriver sürücüleri uygulamada kullanılmıştır. Windows test ortamları için konsept kanıtlama çalışması Windows Server 2012 R2 kullanılarak gerçekleştirilmiş olup bu ortam üzerinde Firefox ve Chrome tarayıcılarına ait belirtilen sürümlerin yanında Internet Explorer 11 tarayıcısı da sunulmuştur.

Farklı bilgisayar ve tarayıcılar için dağıtık test koşturma çalışmaları sırasında Selenium test aracının bazı düğümlerde birtakım metotları gerçekleştiremediği gözlenmiştir. Bu durumun ilk aşamada bir versiyon problemi olduğu düşünüldüğü için çeşitli Selenium sürümlerini (2.46, 2.53, 3.3, 3.4 vd.) [28] kullanarak test koşturulmaya çalışılmış olup denenen bütün sürümlerde aynı düğümlerin aynı metotlarına gelindiğinde testin ilgili düğüm için sürdürülemediği gözlenmiştir. TestNG çatısına ait çeşitli sürümler, tarayıcıların Selenium betiklerini koşturmasını sağlayan sürücülere (geckodriver, chromedriver, iedriver vd.) [28] ait farklı sürümler ve bunların Selenium sürümleri ile kombinasyonları denenerek sorun tespit edilmeye çalışılmış fakat başarılı olunamamıştır.

Kullanılan çözümlere ait sürümlerden kaynaklanmadığı anlaşılan bu hatanın, metotların çalıştığı ve çalışmadığı düğümler karşılaştırıldığında, Java platformlarının kullandığı lokal (java.util.Locale) objelerin seçili dil ve coğrafi ortam parametrelerinin Selenium'un arka planda çalıştırdığı metotlarda sorunlara neden olduğu gözlenmiştir. Özellikle fare ile seçim yapmakta kullanılan "click()" fonksiyonunda gözlenen

problem, Selenium Grid aracının düğüm ve dağıtıcı bağlantısını sağlayan Selenium Server Standalone [28] uygulamasının çalıştırıldığı komuta aşağıda gösterildiği gibi parametre olarak İngilizce dili (en) ve Birleşmiş Milletler (US) coğrafi ortamının eklenmesi ve uygulamanın bu konfigürasyonda çalıştırılması ile aşılmıştır:

-Duser.language=en -Duser.country=US

Windows işletim sistemi tabanlı ortamlar üzerinde test yürütme çalışmaları gerçekleştirilirken, Selenium aracının bazı Windows sürümlerinde platform bilgisi ile otomatik eşleşme sağlayamadığı gözlenmiştir. Windows Server 2016 ile çalışırken karşılaşılmayan bu hata ile Windows Server 2012 R2 sürümünde karşılaşılmıştır. Tarayıcı sürümü ve ortam çözünürlüğü parametrelerinin otomatik olarak algılanmasında yaşanan sorunun çözümünde kullanılan yöntem platform bilgisi için de uygulanmıştır. Selenium Server Standalone uygulaması çalıştırılırken kullanılan komut içerisinde versiyon parametresine tarayıcı sürümü ve ortam çözünürlüğü ile birlikte platform bilgisi de aşağıda gösterildiği gibi eklenmiştir:

-browser browserName=firefox,version=60-1920x1080-Windows2012Server

İşlevsel test tipinde video kayıt özelliği açık kaynak kodlu üçüncül bir yazılım olan Selenium Video Node API'si aracılığıyla gerçekleştirilmiştir [52]. Testler yürütüldüğü sırada video kayıt işleminin otomatik olarak gerçekleştirilmesi için Selenium Video Node ve Selenium Server Standalone uygulamaları, dağıtıcı ve düğümler üzerinde, aşağıdaki komutlarda gösterildiği doğrultuda birlikte başlatılmaktadır:

java -cp
selenium-video-node-2.5.jar:selenium-server-standalone-3.4.0.jar
org.openqa.grid.selenium.GridLauncherV3
-servlets com.aimmac23.hub.servlet.HubVideoDownloadServlet -role hub

java -cp
selenium-video-node-2.5.jar:selenium-server-standalone-3.4.0.jar

```
org.openqa.grid.selenium.GridLauncherV3
-servlets com.aimmac23.node.servlet.VideoRecordingControlServlet
-proxy com.aimmac23.hub.proxy.VideoProxy -role node
```

Selenium Server Standalone ve Selenium Video Node uygulamaları birlikte başlatılırken bu uygulamaların birbirleri ile uyumlu sürümlerinin kullanılmasına dikkat edilmelidir.

BÖLÜM 4. UYGULAMA BULGULARI VE TARTIŞMA

Tasarım modeli ve detayları sunulan bu yaklaşımın öne sürülmesindeki ilk sebep çapraz tarayıcı testlerinin çeşitli ortamlar üzerinde gerçekleştirilmesine duyulan ihtiyaçtır. Ticari ve açık kaynak kodlu araçlar ile otomatize edilmesi mümkün olan bu testleri zorlaştıran en büyük etken test ortamlarının bahsedilen çeşitliliğidir. Bu problemin etkili bir biçimde aşılması için, sunulan tasarım modelinde, bulut bilişim konseptinin getirdiği yenilikler ile bu testlerin otomasyonu işlemleri birleştirilmiştir.

4.1. Özellik ve Yapı Bakımından Karşılaştırma

Günümüzde, bu çalışmada amaçlanan yenilikler doğrultusunda üretilmiş benzer araçlar veya çapraz tarayıcı uyumsuzluklarının tespitinde kullanılan farklı yaklaşımlar mevcuttur. Gerçekleştirilen kaynak taraması ile alanlarının en bilindik örnekleri durumunda bulunan araç ve yaklaşımlar incelenmiş olup bu araç ve yaklaşımlar ile çalışmada sunulan tasarım modeli özellikleri bakımından karşılaştırıldığında Tablo 4.1.'deki sonuç elde edilmiştir.

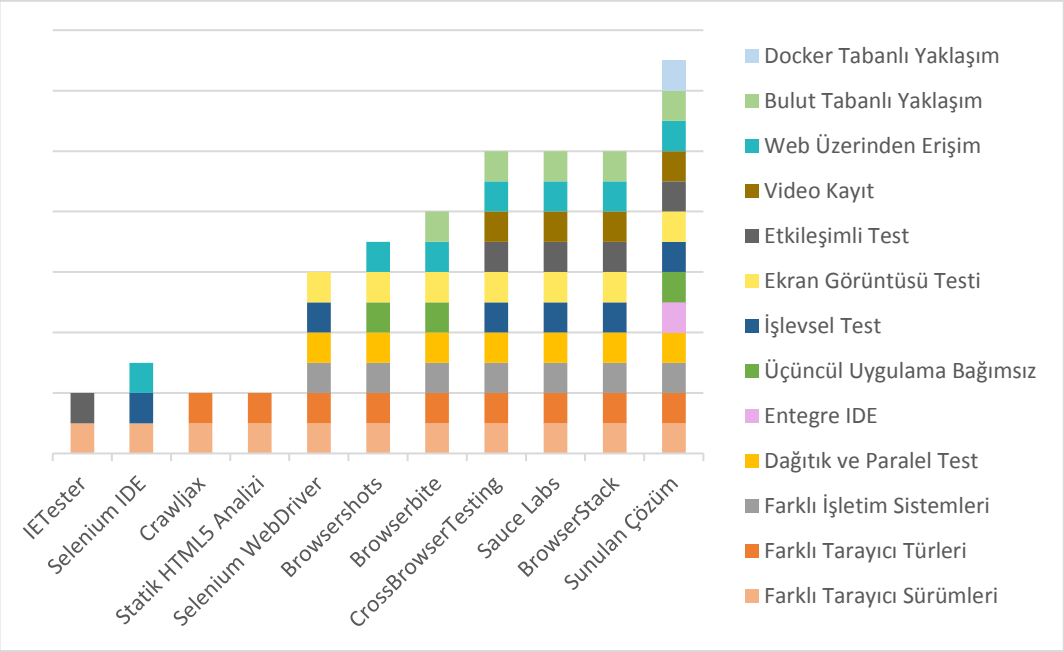
Tablo 4.1. Özellikleri ve yapıları bakımından sunulan çözüm ile mevut araç ve yaklaşımların karşılaştırması

	Referanslar	Farklı tarayıcı sürümleri	Farklı tarayıcı türleri	Farklı işletim sistemleri	Dağıtık ve paralel test	Entegre IDE	Üçüncül uygulama bağımsızlığı	İşlevsel test	Ekran görüntüsü testi	Etkileşimli test	Video kayıt	Web üzerinden erişim	Bulut tabanlı yaklaşım	Docker tabanlı yaklaşım
IETester	[27]	+	-	-	-	-	-	-	-	+	-	-	-	-
Selenium IDE	[28]	+	-	-	-	-	-	+	-	-	-	+	-	-
Crawljax	[30]	+	+	-	-	-	-	-	-	-	-	-	-	-
Statik HTML5 analizi	[32]	+	+	-	-	-	-	-	-	-	-	-	-	-
Selenium WebDriver	[28]	+	+	+	+	-	-	+	+	-	-	-	-	-
Browsershots	[36]	+	+	+	+	-	+	-	+	-	-	+	-	-
Browserbite	[37]	+	+	+	+	-	+	-	+	-	-	+	+	-
CrossBrowserTesting	[38]	+	+	+	+	-	-	+	+	+	+	+	+	-
Sauce Labs	[39]	+	+	+	+	-	-	+	+	+	+	+	+	-
BrowserStack	[40]	+	+	+	+	-	-	+	+	+	+	+	+	-
Sunulan çözüm		+	+	+	+	+	+	+	+	+	+	+	+	+

Tablo 4.1.'de gözlendiği gibi çapraz tarayıcı testi alanındaki araçlar ve yöntemlerle karşılaştırıldığında tasarım modeli sunulan çözümün yetkinlikleri diğer çözümlere farklı açılardan üstünlük göstermektedir. Özellikle, yalnızca bu çözümde bulunan entegre IDE özelliği ve Docker tabanlı yaklaşım yapısı bu çalışmayı mevcut çalışmalardan bir adım ileriye taşımaktadır.

GUI (kullanıcı arayüzü) üzerinden entegre bir IDE sunulması ile kullanıcının test betikleri üzerinde yazma, silme, kaydetme ve derleme işlemlerini üçüncül bir uygulamaya ihtiyaç duymadan gerçekleştirebilmesi sağlanmıştır.

Bu tasarım modelinde, çapraz tarayıcı testine ait bütün alt test tiplerinin hiçbir uygulamaya ihtiyaç duyulmadan tamamen GUI üzerinden ve yalnızca internet bağlantısı sayesinde gerçekleştirilebilmesi öngörülmüştür. Bu durum tasarımı diğer çözümlerin önüne taşıyan bir başka özelliktir. Bu çalışmanın mevcut yaklaşımlara göre özellik ve yapısal bakımdan hangi alanlarda üstünlük sağladığı ve yenilik getirdiği Şekil 4.1.'de toplu olarak gözlenmektedir. Bu özelliklerden en önemlisi, çapraz tarayıcı testlerine yeni bir perspektif sağlaması bakımından, çalışmada Docker tabanlı bir yaklaşım tasarlanması ve geliştirilmesidir.



Şekil 4.1. Çapraz tarayıcı testi çözümlerine genel bakış

4.2. Docker Tabanlı Yapının Verimlilik Değerlendirmesi

Çapraz tarayıcı testlerinin otomasyonunda bulut tabanlı çözümlerin klasik çözümlere göre test ortamlarının oluşturulması ve sunulması bakımından üstünlükleri bulunmaktadır. Önerilen tasarım modelinde, bulut tabanlı yaklaşımın Docker teknolojisi ile güçlendirilerek test ortamlarının oluşturulması ve izolasyonu, kaynak kullanımı gibi işlemlerde bulut tabanlı çözümlere üstünlük sağlaması öngörülmektedir.

Docker teknolojisi ile birlikte kullanılan bulut tabanlı yaklaşımın, salt bulut tabanlı yaklaşıma göre test süreleri bakımından veriminin değerlendirilmesi için, tasarıma göre geliştirilen uygulama ile bulut tabanlı çözümlerin önde gelen bir örneği olan CrossBrowserTesting aracı arasında işlevsel test tipi baz alınarak karşılaştırma yapılmıştır.

Uygulamanın üzerinde çalışacağı bulut altyapısı olarak OpenStack Pike [48] sürümü kullanılmıştır. Docker kullanımı için bulut makineleri üzerinde Docker 18.03.1-ce [24] sürümünün kurulumu gerçekleştirilmiştir.

Docker kullanımının çapraz tarayıcı testine süre olarak etkisini gözlemleyebilmek için geliştirilen uygulamada Linux tabanlı ortamlar tercih edilmiştir. Ancak CrossBrowserTesting uygulamasında Linux işletim sistemi platform olarak sunulmadığı için bu aracı kullanılırken Windows işletim sistemi test ortamı olarak tercih edilmiştir. Bu değerlendirmede, test ortamlarının sayısı doğrultusunda test sürelerine dair karşılaştırma gerçekleştirilmiş olup test ortamı tipleri önem arz etmemektedir. Buna rağmen kesin bir yargıya varabilmek adına geliştirilen uygulamada Windows ve Linux test ortamları arasında da kıyaslama gerçekleştirilmiştir.

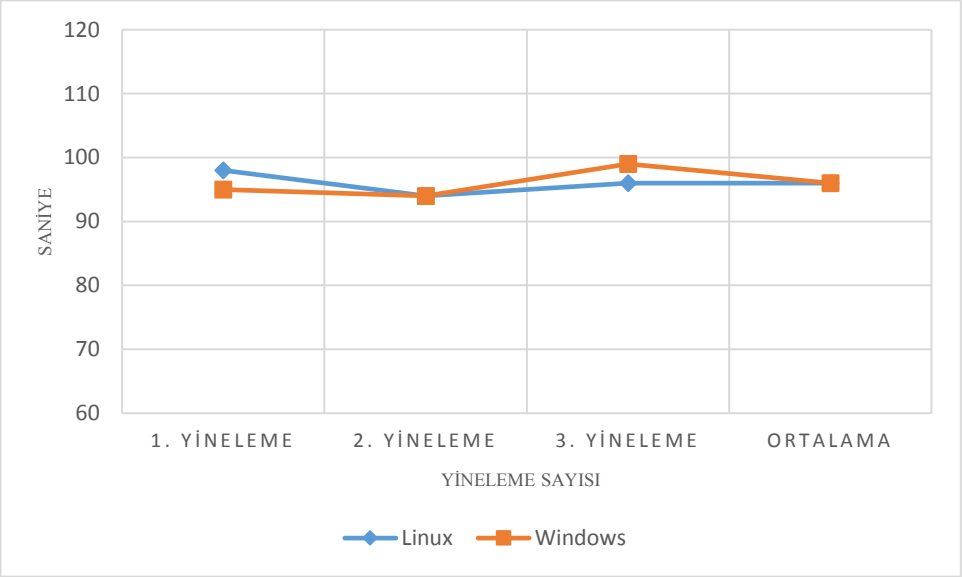
Karşılaştırmalar CrossBrowserTesting aracında Windows 8 işletim sistemi, geliştirilen uygulamada ise Linux işletim sistemi (Ubuntu 14.04 Docker konteyneri) kullanılarak gerçekleştirilmiştir. Her iki araç üzerinde Chrome tarayıcısının 63, Firefox tarayıcısının ise 56, 57 ve 58 sürümlerine çeşitli kombinasyonlar tercih edilmiş olup çözünürlük olarak 1920x1080 kullanılmıştır. Bunun yanı sıra daha kesin değerler elde etmek adına test yürütme işlemi üç kez yinelenerek ortalama süre hesaplanmıştır.

Yapılan değerlendirmeler Şekil 4.2.'de verilen, Java ile yazılmış Selenium test betiği yürütülerek gerçekleştirilmiştir.

```
driver.navigate().to("http://www.tubitak.gov.tr");
Thread.sleep(10000);
String xPath = "//a[@href='/tr/icerik-baskanlik-birimleri']";
WebDriverWait wait = new WebDriverWait(driver, 30);
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(xPath)));
WebElement element = driver.findElement(By.xpath(xPath));
wait.until(ExpectedConditions.elementToBeClickable(element));
element.click();
Thread.sleep(10000);
assertTrue(driver.getTitle().contains("Birimler"));
xPath = "//a[@href='/tr/icerik-merkez-ve-enstituler']";
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(xPath)));
element = driver.findElement(By.xpath(xPath));
wait.until(ExpectedConditions.elementToBeClickable(element));
element.click();
Thread.sleep(10000);
assertTrue(driver.getTitle().contains("Merkez"));
xPath = "//a[@href='/tr/icerik-e-TUBITAK']";
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(xPath)));
element = driver.findElement(By.xpath(xPath));
wait.until(ExpectedConditions.elementToBeClickable(element));
element.click();
Thread.sleep(10000);
assertTrue(driver.getTitle().contains("E-TÜBİTAK"));
xPath = "//a[@href='/tr/haber-arsivi']";
wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(xPath)));
element = driver.findElement(By.xpath(xPath));
wait.until(ExpectedConditions.elementToBeClickable(element));
element.click();
Thread.sleep(10000);
assertTrue(driver.getTitle().contains("Haber"));
driver.quit();
```

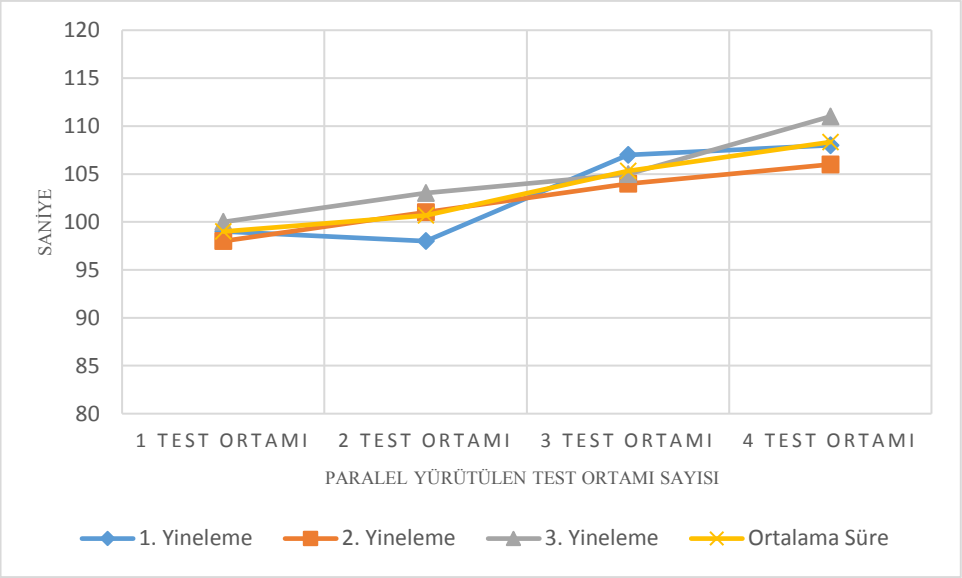
Şekil 4.2. TÜBİTAK web sayfası için örnek Selenium test betiği

İlk olarak, geliştirilen uygulama üzerinde idle (çalışır ve boş) durumda bulunan Windows tabanlı bir bulut makinesi ile Linux tabanlı test ortamı karşılaştırılarak bu test ortamlarına ait sonuçların birbirlerine yakın sürelerde üretildiği tespit edilmiştir. Şekil 4.3.'te elde edilen sonucun gösterildiği bu gözlem, CrossBrowserTesting uygulamasında herhangi bir Linux platformu test ortamı olarak sunulmadığı için gerçekleştirilmiştir.



Şekil 4.3. Geliştirilen uygulamada Linux ve Windows test ortamlarına ait çalışma sürelerinin karşılaştırması

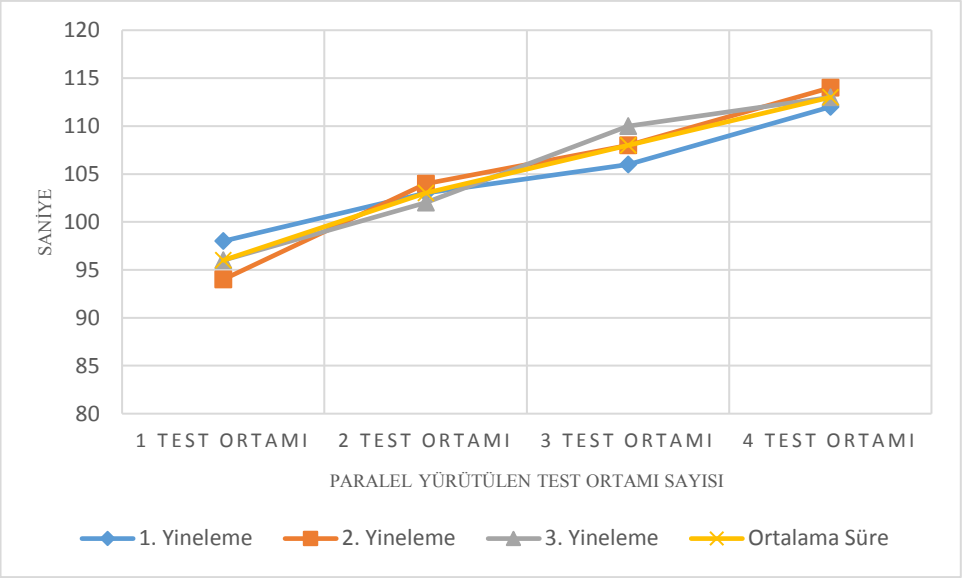
Geliştirilen uygulama kullanılarak Linux ve Windows ortamlarında test yürütülürken, testlerin sonuçlanma süreleri arasında kayda değer farklar oluşmadığı ve ortalama sürelerin denk olduğu gözlenmiştir. Bu gözlemin ardından, her iki uygulamada yukarıda sıralanan test ortamları kullanılarak paralel testler gerçekleştirilmiştir. Elde edilen sonuçlar CrossBrowserTesting uygulaması için Şekil 4.4.'te gösterilmektedir.



Şekil 4.4. CrossBrowserTesting paralel test süreleri

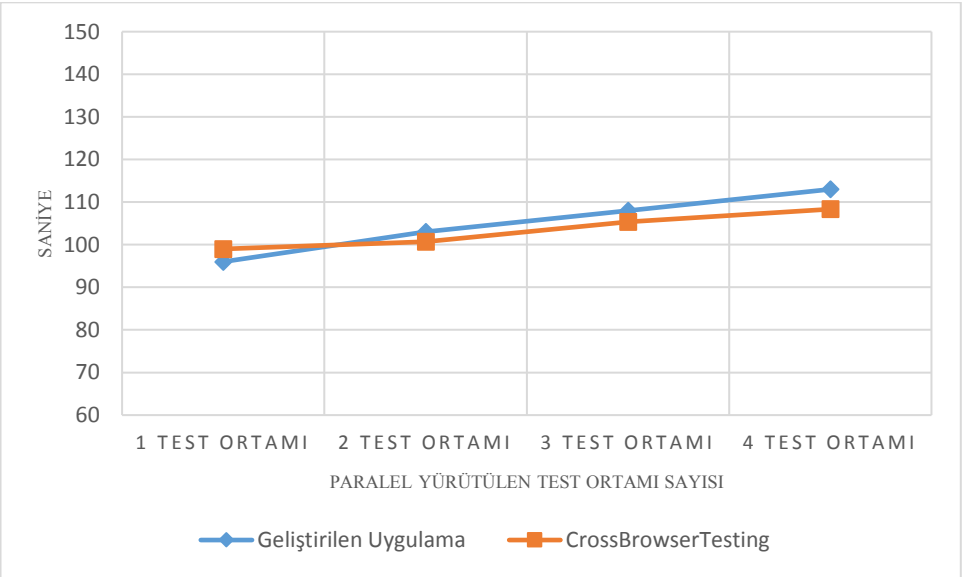
Şekil 4.4.'te görüldüğü gibi bulut tabanlı bir test aracı olan CrossBrowserTesting üzerinde 1 ila 4 test ortamı kullanılarak gerçekleştirilen testlerin sonuçlanma süresi ortalama 100-110 saniye bandına yerleşmektedir. CrossBrowserTesting aracı açık kaynak kodlu bir yazılım olmadığı ve tasarımına dair erişime açık herhangi bir kaynak bulunmadığı için arka planda uygulanan algoritmalar hakkında kesin bir bilgi elde edilememektedir. Ancak bu gözlemler ile iki aracın algoritmik olarak karşılaştırılması hedeflenmemiş olup Docker tabanlı yaklaşımın çalışma süreleri bakımından, bulut tabanlı bir araca göre durumunun değerlendirilmesi amaçlanmıştır.

Aynı Selenium test betiği, geliştirilen uygulamada Linux tabanlı 1 ila 4 test ortamı kullanılarak yürütülmüş olup Docker tabanlı yaklaşım için elde edilen sonuçlar Şekil 4.5.'te gösterildiği gibidir.



Şekil 4.5. Geliştirilen uygulamada elde edilen paralel test süreleri

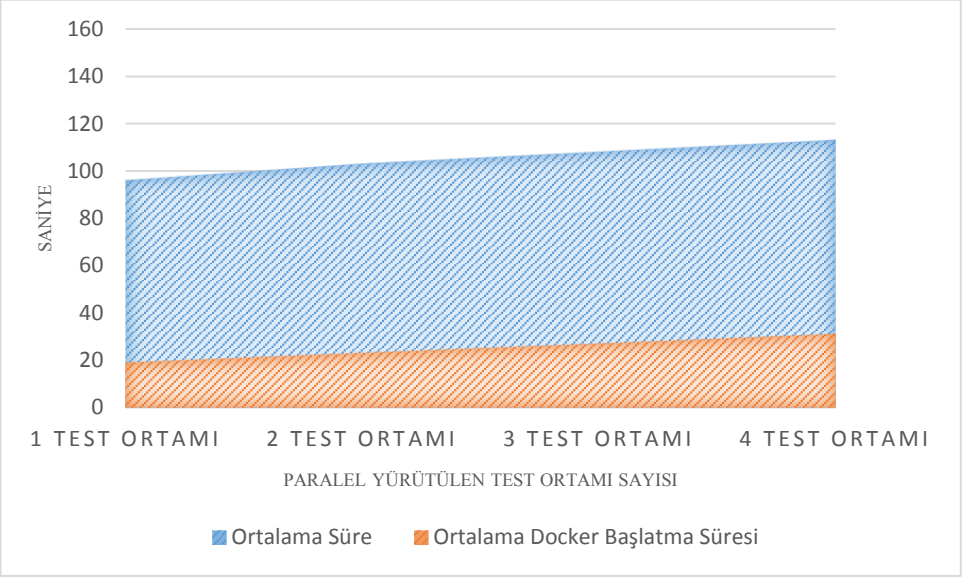
Şekil 4.5.'te verilen grafikte görüldüğü gibi bu uygulamanın verilen test betiği için çalışma süresi ortalama 95-115 saniye bandına yerleşmektedir. Bu iki değerlendirme ile çizilen grafiklerdeki ortalama süre verileri kullanılarak CrossBrowserTesting ve geliştirilen uygulama arasındaki kıyaslama Şekil 4.6.'da gösterildiği gibi gerçekleştirilmiştir.



Şekil 4.6. Docker tabanlı ve bulut tabanlı iki yaklaşımın karşılaştırması

Şekil 4.6.'da verilen grafik doğrultusunda iki test yaklaşımı için test yürütme süreleri arasında kayda değer farkların oluşmadığı sonucuna varılmaktadır. Bu bölümde yapılan değerlendirmelerle gösterilmek istenen durum Docker tabanlı yaklaşım ile bulut tabanlı yaklaşım arasında süre bazında verimliliği etkileyecek herhangi bir farkın oluşmadığıdır.

Bulut tabanlı yaklaşım ile karşılaştırması yapılan uygulama kendi içerisinde değerlendirilerek Docker konteynerlerinin başlatılma ve test yürütme işlemlerinin gerçekleştirilme süreleri artan test ortamı sayısı doğrultusunda incelenmiştir. Bu inceleme sonucunda Şekil 4.7.'de verilen grafik elde edilmiştir. Grafikte de görüldüğü gibi, artan test ortamı doğrultusunda Docker konteynerlerin test süreci içerisindeki oluşum süresi göz ardı edilebilecek bir artış ile 20 saniye bandında yer almaktadır.



Şekil 4.7. Docker konteyner oluşturma ve test sürelerinin karşılaştırması

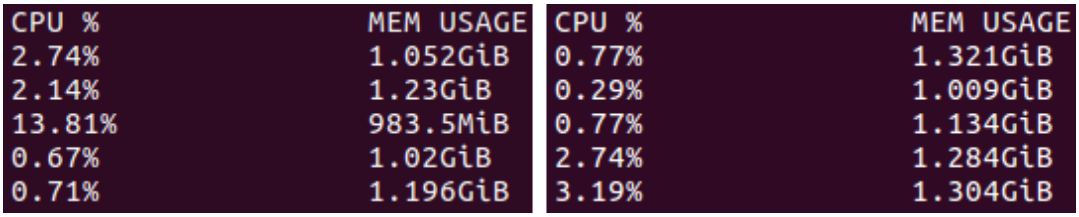
Docker konteynerlerinin başlatılması çok kısa süren bir işlem olup bu değerlendirmede elde edilen süreler konteyner içerisinde çalıştırılan Agent ve VNC uygulamalarından kaynaklanmaktadır.

Docker kullanılan bulut tabanlı çapraz tarayıcı test otomasyonu çözümü, salt bulut tabanlı çözümler ile test süreleri bakımından yaklaşık olarak denk durumda gözükmektedir. Test süresi bakımından herhangi bir dezavantaja sebep olmayan

Docker tabanlı test ortamlarının tercih edilmesi, özellikle kaynak kullanımında önemli avantajlar sağlamaktadır.

Hipervizörler (hypervisor) sanallaştırma platformunu yönetirken sanal makine ve üzerinde bulunduğu fiziksel sunucu arasına katmanlar ekler [53]. İşletim sistemi, disk, RAM gibi kaynaklar bu sayede fiziki sunucudan yalıtılmış olur ancak performans ve verimlilik açısından bakıldığında konteyner tabanlı sistemler çok daha iyi bir çözüm sunmaktadır [53–55].

Gerçekleştirilen karşılaştırmalar sırasında, geliştirilen uygulama üzerinde dört test ortamı seçilerek yapılan gözlem için, testler yürütüldüğü esnada konteynerlerin iki farklı ana ait kaynak tüketim verileri Şekil 4.8.'de gösterilmiştir.



Şekil 4.8. Docker konteynerlerinin test yürütme işlemi sırasında iki farklı ana ait kaynak tüketim verileri

Konteynerler oluştuktan sonra gerekli uygulamaların başlatılması ve testlerin yürütülmesi süreci boyunca elde edilen ortalama değerler Tablo 4.2.'de gösterilmiştir.

Tablo 4.2. Docker konteynerlerinin test yürütme işlemi sırasında ortalama kaynak tüketim değerleri

CPU %	Bellek kullanımı
%0 - %398	900 MB – 1,35 GB

CPU %: %100 değeri 1 CPU'nun tam olarak kullanımı manasına gelmektedir.

Şekil 4.8. ve Tablo 4.2.'de verilen değerler de göstermektedir ki uygulamaların çalışması esnasında CPU ve bellek kullanım ihtiyaçları anlık olarak değişmektedir. Konteynerler ile birlikte Agent ve VNC uygulamalarının başlatılması sırasında düşük bellek ve yüksek CPU kullanımı gözlenmiş olup, testler yürütülürken ise bellek değerleri artarak CPU kullanımı çok düşük seviyelere inmiştir. Bu veriler

desteklemektedir ki test ortamlarına ayrılan kaynakların ortaklaşa kullanımı altyapıyı daha verimli tüketme olanağı sağlamaktadır. CPU kullanımında gözlenen 0 ila 398 arası geniş değişim aralığı, kaynakların ortaklaşa kullanılmaması durumunda, işlem hızlarının düşmesi ya da gereksiz yere CPU işgali ihtimaline işaret etmektedir.

Konteyner tabanlı sanallaştırmada bir işletim sisteminin örneklenerek çoğaltılması tek bir çekirdek üzerinden gerçekleştirilmektedir ve işlevsel tekrarlar olmamaktadır [55]. Bu durum konteynerlere hız, hafiflik ve daha mobil bir yapı kazandırmaktadır. Dolayısıyla kaynak kullanımı klasik sanallaştırma yöntemlerine nazaran daha iyi bir seviyeye taşınmaktadır.

Preeth ve diğerlerine göre bulut sistemleri sanal makine tabanlı çalışmakta olup sanal makine performansı bütün bir bulut performansını etkilemektedir ve konteyner tabanlı sanallaştırma bu yapıya bir alternatif olmuştur [56]. Bulut altyapı sağlayıcıları genel olarak sanal makineleri idle konumunda hazır bekletmekte olup bu durum gereksiz kaynak kullanımına sebep olmaktadır. Bununla birlikte bir fiziksel sunucu üzerinde sanal makineye kıyasla ortak kaynak kullanımından ve düşük kaynak tüketiminden dolayı daha fazla konteyner çalıştırılabileceği için daha yüksek bir kaynak tasarrufu sağlanmış olur [55]. Konteynerler üzerinde gerçekleştirilecek kapatıp-açma hatta başlatma ve sonlandırma işlemleri sanal makineler ile kıyaslandığında çok daha hızlı gerçekleşmektedir [56]. Dolayısıyla hızlı hizmet sunmak adına sanal makinelerin idle konumda bekletilmesi gerekirken konteynerler için böyle bir ihtiyaç söz konusu değildir.

Konteynerler önceden derlenmiş olan Docker imajları kullanılarak başlatılırlar. Docker imajları katmanlar halinde derlenir ve yeni bir imaj derlenirken işlemin gerçekleştirildiği makine üzerinde bulunan imaj katmanları incelenerek derleme eksik katmandan başlatılır. Dolayısıyla derlenmiş bir Dockerfile'ın son satırında yapılan değişiklik sonrası derleme işlemi yeniden gerçekleştirildiğinde yalnızca son satırda yapılan değişiklik derlenerek disk kullanımı düşürülmüş olur. Derlenen imajlar kolaylıkla taşınabilir veya havuza kaydedilebilir ve taşındığı yerde Docker haricinde herhangi bir bağımlılık olmadan kolaylıkla çalıştırılabilir.

Günümüzde yetersiz kalan klasik sunucu yöntemlerinin yerini alan bulut tabanlı sanallaştırma çözümleri gün geçtikçe yerini Docker gibi konteyner sistemleri ile geliştirilmiş yöntemlere bırakmaktadır. Klasik yöntemlere göre bulut tabanlı test otomasyonu esnek bir yapı sunarken, Docker ile desteklenmiş bulut tabanlı çözüm daha esnek ve mobil bir yapı sağlamakta olup bulut altyapısının daha verimli kullanılmasına imkân tanımaktadır.

BÖLÜM 5. SONUÇ VE ÖNERİLER

Bulut tabanlı çapraz tarayıcı test otomasyonu uygulaması için özgün bir tasarım modeli sunulan bu çalışmada bulut bilişim teknolojisinin esneklik ve dinamik kaynak kullanımı gibi özelliklerinin bu testlerin doğasına çok uygun imkanlar sunduğu gözlenmiştir. Platform çeşitliliğinin ve ortam yalıtımının önemli olduğu bu test uygulamaları için klasik sunucu yöntemleri yerine bulut alt yapı kullanımı maliyet tasarrufu sağladığı kadar kaynakları daha verimli kullanma imkânı sunduğu için kaynak tasarrufu da sağlamaktadır. Esnek yapısı sayesinde kullanıcı tarafında oluşacak bekleme süreleri minimum seviyelere çekilebilmektedir.

Çalışmada Docker kullanımının bu faydaları bir adım daha ileri taşıdığı gözlenmiştir. Her ne kadar Windows ortamlar için bu testlere uygun olacak doğrultuda Docker imajları oluşturmak mümkün olmasa bile, ortam olarak Windows tercih edildiğinde dahi Docker teknolojisinden fayda sağlanmıştır. Dağıtıcı görevi üstlenecek makine için bu tasarımda her zaman Docker konteynerleri tercih edilmektedir. Bu sayede Windows makinelerin hem dağıtıcı görevi üstlenerek boşa kullanımının önüne geçilmiş hem de düğüm ve dağıtıcı görevlerini bir arada üstlenmesi ihtimali engellendiği için ekstra yükten kurtarılmıştır.

Gelecekte Docker ve Windows uyumu ile ilgili gelişmeler doğrultusunda Windows sürümleri de Docker teknolojisi kullanılarak üretilebilir ve tasarımın daha verimli yapıya kavuşması sağlanabilir.

Çalışmaya görüntü işleme algoritmaları ile geliştirilecek bir karşılaştırma servisi eklenebilir ve bu karşılaştırmada öğrenme tekniklerinden faydalanılarak her defasında

daha doğru sonuç üretimi sağlanabilir. Ayrıca tercih edilen ortamlarda test yürütülmesinin ardından alınan hatalar doğrultusunda yine öğrenme teknikleri kullanılarak hata ile karşılaşma ihtimalinin olduğu başka test ortamlarının tespiti yapılabilir ve bu doğrultuda direktifler sunulabilir.

KAYNAKLAR

[1] Whittaker, J. A., What is software testing? And why is it so hard?, *IEEE Softw.*, c. 17, sayı 1, ss. 70–79, 2000.

[2] Myers, G., *The Art of Software Testing, Second edition*, c. 15. 2004.

[3] Sheehan, K. M. ve Young, A. T., It's a Small World After All: Internet Access and Institutional Quality, *Contemp. Econ. Policy*, c. 33, sayı 4, ss. 649–667, 2015.

[4] Katherine, A. V ve Alagarsamy, D. K., Conventional Software Testing Vs. Cloud Testing, *Int. J. Sci. ...*, c. 3, sayı 9, ss. 1–5, 2012.

[5] Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., ve Rabkin, A., A view of cloud computing, *Commun. ACM*, c. 53, sayı 4, s. 50, 2010.

[6] Inki, K., Ari, I., ve Sozer, H., A Survey of Software Testing in the Cloud, *2012 IEEE Sixth Int. Conf. Softw. Secur. Reliab. Companion*, ss. 18–23, 2012.

[7] Harikrishna, P. ve Amuthan, A., A survey of testing as a service in cloud computing, *6th Int. Conf. Comput. Commun. Informatics, ICCCI 2016*, 2016.

[8] Hayes, B., Cloud computing, *Commun. ACM*, c. 51, sayı 7, s. 9, 2008.

[9] Gong, C., Liu, J., Zhang, Q., Chen, H., ve Gong, Z., The characteristics of cloud computing, *Proc. Int. Conf. Parallel Process. Work.*, ss. 275–279, 2010.

[10] Mell, P. ve Grance, T., The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology, *Natl. Inst. Stand. Technol. Inf. Technol. Lab.*, c. 145, s. 7, 2011.

[11] Kim, W., Cloud computing: Today and tomorrow, *J. Object Technol.*, c. 8, sayı 1, ss. 65–72, 2009.

[12] Rimal, B. P., Choi, E., ve Lumb, I., A taxonomy and survey of cloud computing systems, *NCM 2009 - 5th Int. Jt. Conf. INC, IMS, IDC*, ss. 44–51, 2009.

[13] Turner, M., Budgen, D., ve Brereton, P., Turning software into a service, Comput-er, Vol. no. 10, ., c. 36 SRC-, ss. 38–44, 2003.

[14] Lawton, G., Developing Software Online with Platform-as-a- Service Technology, *Computer (Long. Beach. Calif).*, c. 41, sayı 6, ss. 13–15, 2008.

[15] Bhardwaj, S., Jain, L., ve Jain, S., Cloud Computing : a Study of Infrastructure As a Service (Iaas), *Int. J. Eng.*, c. 2, sayı 1, ss. 60–63, 2010.

[16] Yu, L., Tsai, W. T., Chen, X., Liu, L., Zhao, Y., Tang, L., ve Zhao, W., Testing as a service over cloud, *Proc. - 5th IEEE Int. Symp. Serv. Syst. Eng. SOSE 2010*, ss. 181–188, 2010.

[17] Katherine, a V. ve Alagarsamy, K., Software Testing in Cloud Platform : A Survey, *Int. J. Comput. Appl.*, c. 46, sayı 6, ss. 21–25, 2012.

[18] Gao, J., Bai, X., Tsai, W. T., ve Uehara, T., Testing as a service (TaaS) on clouds, *Proc. - 2013 IEEE 7th Int. Symp. Serv. Syst. Eng. SOSE 2013*, ss. 212–223, 2013.

[19] Gao, J., Bai, X., ve Tsai, W., Cloud Testing - Issues, Challenges, Needs and Practice, *Softw. Eng. An Int. J.*, c. 1, sayı 1, ss. 9–23, 2011.

[20] Leung, H. K. N. ve White, L., Insights into Regression Testing, *Proc. Int. Conf. Softw. Maint.*, ss. 60–69, 1989.

[21] Gupta, R., Harrold, M. J., ve Soffa, M. L., An approach to regression testing using slicing, *Softw. Maintenance, 1992. Proceerdings., Conf.*, sayı November, ss. 299–308, 1992.

[22] Choudhary, S. R., Prasad, M. R., ve Orso, A., X-PERT: Accurate identification of cross-browser issues in web applications, *Proc. - Int. Conf. Softw. Eng.*, ss. 702–711, 2013.

[23] Mesbah, A. ve Prasad, M. R., Automated cross-browser compatibility testing, *2011 33rd Int. Conf. Softw. Eng.*, ss. 561–570, 2011.

[24] Docker - Build, Ship, and Run Any App, Anywhere, https://www.docker.com/, Erişim Tarihi: 10.07.2018.

[25] Manhas, J., Comparative Study of Cross Browser Compatibility as Design Issue in Various Websites, *BIJIT - BVICAM's Int. J. Inf. Technol.*, c. 7, ss. 815–820, 2015.

[26] Kaalra, B. ve Gowthaman, K., Cross Browser Testing Using Automated Test Tools, *Int. J. Adv. Stud. Comput. Sci. Eng.*, c. 3, sayı 10, s. 7, 2014.

[27] Browser Compatibility Check for Internet Explorer Versions from 5.5 to 11., https://www.my-debugbar.com/wiki/IETester/HomePage, Erişim Tarihi: 10.07.2018.

[28] Selenium - Web Browser Automation, https://www.seleniumhq.org/, Erişim Tarihi: 10.07.2018.

[29] Holmes, A. ve Kellogg, M., Automating Functional Tests Using Selenium, *Agil. 2006*, ss. 270–275, 2006.

[30] Crawling Ajax-based Web Applications, http://crawljax.com/, Erişim Tarihi: 10.07.2018.

[31] Shi, H. ve Zeng, H., Cross-Browser Compatibility Testing Based on Model Comparison, *2015 Int. Conf. Comput. Appl. Technol.*, ss. 103–107, 2015.

[32] Xu, S. ve Zeng, H., Static analysis technique of cross-browser compatibility detecting, *Proc. - 3rd Int. Conf. Appl. Comput. Inf. Technol. 2nd Int. Conf. Comput. Sci. Intell. ACIT-CSI 2015*, ss. 103–107, 2015.

[33] Rahman, M. F. 2014. Browser-Based Automation Testing Using Selenium Webdriver. Turku Üniversitesi, Uygulamalı Bilimler Tezi

[34] TestNG, http://testng.org/doc/, Erişim Tarihi: 10.07.2018.

[35] Bindal, P. ve Gupta, S., Test Automation Selenium WebDriver using TestNG, c. 3, sayı 9, ss. 18–40, 2014.

[36] Check Browser Compatibility, Cross Platform Browser Test - Browsershots, http://browsershots.org/, Erişim Tarihi: 10.07.2018.

[37] Browserbite - Automatic cross browser testing, http://browserbite.com/, Erişim Tarihi: 10.07.2018.

[38] Cross Browser Testing Tool: 1500+ Real Browsers & Devices, https://crossbrowsertesting.com/, Erişim Tarihi: 10.07.2018.

[39] Sauce Labs: Cross Browser Testing, Selenium Testing, and Mobile Testing, https://saucelabs.com/, Erişim Tarihi: 10.07.2018.

[40] Cross Browser Testing Tool. 1000+ Browsers, Mobile, Real IE, https://www.browserstack.com/, Erişim Tarihi: 10.07.2018.

[41] Dallmeier, V., Pohl, B., Burger, M., Mirold, M., ve Zeller, A., WebMate: Web Application Test Generation in the Real World, *2014 IEEE Seventh Int. Conf. Softw. Testing, Verif. Valid. Work.*, ss. 413–418, 2014.

[42] noVNC, http://novnc.com/, Erişim Tarihi: 10.07.2018.

[43] Maven, https://maven.apache.org/, Erişim Tarihi: 10.07.2018.

[44] Fredrich, T., RESTful Service Best Practices: recommendations for creating web services, ss. 1–34, 2013.

[45] Webb, P., Syer, D., Long, J., Nicoll, S., Winch, R., Wilkinson, A., Overdijk, M., Dupuis, C., Deleuze, S., Simons, M., Pavi, V., Bryant, J., ve Bhave, M., Spring Boot Reference Guide, 2018.

[46] Singla, S. ve Kaur, H., Selenium Keyword Driven Automation Testing Framework, *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, c. 4, sayı 6, ss. 2277–128, 2014.

[47] AssertThat, https://www.assertthat.com/, Erişim Tarihi: 10.07.2018.

[48] Open source software for creating private and public clouds, https://www.openstack.org/, Erişim Tarihi: 10.07.2018.

[49] Beernaert, L., Matos, M., Vilaça, R., ve Oliveira, R., Automatic elasticity in OpenStack, *Proc. Work. Secur. Dependable Middlew. Cloud Monit. Manag. - SDMCMM '12*, sayı December, ss. 1–6, 2012.

[50] Rosado, T. ve Bernardino, J., An overview of openstack architecture, *Acm*, ss. 366–367, 2014.

[51] Microsoft - Docker Hub, https://hub.docker.com/u/microsoft/, Erişim Tarihi: 10.07.2018.

[52] Aimmac23/selenium-video-node: Code to add video recording capability to Selenium Nodes, https://github.com/aimmac23/selenium-video-node, Erişim Tarihi: 10.07.2018.

[53] Hwang, J., Zeng, S., ve Wood, T., A component-based performance comparison of four hypervisors, *Integr. Netw. Manag. ...*, sayı May 2014, ss. 269–276, 2013.

[54] Sefraoui, O., Aissaoui, M., ve Eleuldj, M., OpenStack: Toward an Open-Source Solution for Cloud Computing, *Int. J. Comput. Appl.*, c. 55, sayı 03, ss. 38–42, 2012.

[55] Singh, S. ve Singh, N., Containers & Docker: Emerging roles & future of Cloud technology, in Proceedings of the 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology, iCATccT 2016, 2017, 804–807.

[56] Preeth, E. N., Mulerickal, J. P., Paul, B., ve Sastri, Y., Evaluation of Docker containers based on hardware utilization, *2015 Int. Conf. Control. Commun. Comput. India, ICCC 2015*, sayı November, ss. 697–700, 2016.

ÖZGEÇMİŞ

Mehmet Emin Küçüker, 13.10.1990 tarihinde Ankara'da doğdu. İlk, orta ve lise eğitimini Ankara'da tamamladı. 2008 yılında Ankara Atatürk Anadolu Lisesi'nden mezun oldu. 2008 yılında başladığı Orta Doğu Teknik Üniversitesi Bilgisayar Mühendisliği Bölümü'nü 2014 yılında tamamladı. 2016 yılında Sakarya Üniversitesi'nde Bilgisayar ve Bilişim Mühendisliği yüksek lisans eğitimine başladı. 2015 yılından bu yana TÜBİTAK Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezinde Araştırmacı olarak görev yapmaktadır.