FEN BİLİMLERİ ENSTİTÜSÜ

Mikroservis Mimarilerinin Güvenliği ve Dayanıklılığı İçin Senaryo Tabanlı Kaos Deneylerinin Simülasyonu

Aybüke ERGÜL

Danışman
Doç. Dr. Mustafa Cem KASAPBAŞI

İkinci Tez Danışmanı
Prof. DR. Rıfat YAZICI

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ
İSTANBUL - 2024

KABUL VE ONAY SAYFASI

Aybüke ERGÜL tarafından hazırlanan **"Mikroservis Mimarilerinin Güvenliği ve Dayanıklılığı İçin Senaryo Tabanlı Kaos Deneylerinin Simülasyonu"** adlı tez çalışması 03/09/2024 tarihinde aşağıdaki jüri üyeleri önünde başarı ile savunularak, İstanbul Ticaret Üniversitesi Fen Bilimleri Enstitüsü **Yüksek Lisans Bilgisayar Mühendisliği Anabilim Dalı**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman	**Unvanı Adı SOYADI** İstanbul Ticaret Üniversitesi	.....
Jüri Üyesi	**Unvanı Adı SOYADI** İstanbul Ticaret Üniversitesi	.....
Jüri Üyesi	**Unvanı Adı SOYADI** İstanbul Ticaret Üniversitesi	.....
Jüri Üyesi	**Unvanı Adı SOYADI** İstanbul Ticaret Üniversitesi	.....
Jüri Üyesi	**Unvanı Adı SOYADI** İstanbul Ticaret Üniversitesi	.....

Onay Tarihi :

Prof. Dr. Necip ŞİMŞEK
Enstitü Müdürü

AKADEMİK VE ETİK KURALLARA UYGUNLUK BEYANI

İstanbul Ticaret Üniversitesi, Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversitede veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

Tarih: 15.08.2024

İmza:

Aybüke ERGÜL

İÇİNDEKİLER

	Sayfa
İÇİNDEKİLER	i
ÖZET	iii
ABSTRACT	iv
TEŞEKKÜR	v
1. GİRİŞ	1
2. LİTERATÜR ÖZETİ	3
2.1. Hata Enjeksiyonu	3
2.2. Senaryo Tabanlı Dayanıklılık Testi	4
2.3. Mikroservisler ve Bulut Mimarisinde Simülasyon	6
3. MATERYAL VE METODOLOJİ	8
3.1. Kaos Mühendisliği Araçlarının Seçimi	8
3.2. Küme ve Mikroservisler Genel Bakışı	9
3.3. Kaos Mühendisliği Uygulaması	10
3.3.1. Senaryo Tasarımı ve Seçimi	11
3.3.2. Kaos Senaryolarının Yürütülmesi	13
3.4. İzleme ve Analiz	14
4. ARAŞTIRMA BULGULARI VE TARTIŞMA	16
5. SONUÇ VE ÖNERİLER	19
KAYNAKLAR	21
ÖZGEÇMİŞ	23

ÖZET

Yüksek Lisans Tezi

MİKROSERVİS MİMARİLERİNİN GÜVENLİĞİ VE DAYANIKLILIĞI İÇİN SENARYO TABANLI KAOS DENEYLERİNİN SİMÜLASYONU

Aybüke ERGÜL

İstanbul Ticaret Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği

Danışman: Doç. Dr. Mustafa Cem KASAPBAŞI

Eş Danışman: Prof. Dr. Rıfat YAZICI
2024, 30

Mikroservis tabanlı mimarilerin popülaritesi arttıkça, sistemlerin dayanıklılığını test etme ihtiyacı da önemli hale gelmiştir. Güvenlik, ölçeklenebilirlik ve performans gibi temel kavramları derinlemesine inceleyen birçok çalışma yapılmıştır. Ancak, teknolojinin hızlı gelişimi, bu mimarilerin güvenilirliğini sağlamada zorlukları beraberinde getirmiştir. Bu zorlukların üstesinden gelmek için kaos mühendisliği gibi yenilikçi yöntemler giderek daha önemli hale gelmektedir.

Kaos mühendisliği, sistemin dayanıklılığını test etmek ve olası zayıf noktaları belirlemek amacıyla sistemin farklı bölümlerine kasıtlı olarak hatalar veya kesintiler getirme yöntemidir. Bu yöntem, özellikle karmaşık bağlantılara sahip mimariler için vazgeçilmezdir. Bu tez, mikroservis mimarileri için kaos mühendisliğinin önemini vurgulamakta ve kaos deneyleri için yaygın olarak kullanılan araçlar ve teknolojilere genel bir bakış sunmaktadır. Ayrıca, kaos deneylerinin planlanması ve yürütülmesi için en iyi uygulamaları özetlemekte ve kaos mühendisliğinin pratikte kullanımının faydalarını ve zorluklarını tartışmaktadır. Kaos deneylerini gerçekleştirmek için mikroservislerle birlikte kümenin olduğu bir proje test ortamı olarak kullanılmıştır. Pod tabanlı, Ağ tabanlı ve Stres tabanlı deneyler gibi çeşitli kaos deney senaryoları, Chaos Mesh ve Litmus gibi açık kaynak yazılımlar kullanılarak projeye uygulanmıştır.

Anahtar Kelimeler: Dayanıklılık, Hata Toleransı, Kaos Mühendisliği, Pod, Küme, Kubernetes, Mikroservisler, Simülasyon, Üretim, Yeniden Yapılanma

ABSTRACT

M.Sc. Thesis

SIMULATING SCENARIO-BASED CHAOS EXPERIMENTS FOR SECURITY AND RESILIENCY OF MICROSERVICES ARCHITECTURES

Aybüke ERGÜL

İstanbul Commerce University
Graduate School of Applied and Natural Sciences
Department of Computer Engineering

Supervisor: Doç. Dr. Mustafa Cem KASAPBAŞI

Co-Supervisor: Prof. Dr. Rıfat YAZICI
2024, 30

As the popularity of microservices-based architectures increases, the need to test the resilience of systems has also become important. Many studies have been conducted that examine in depth basic concepts such as security, scalability and performance. However, the rapid development of technology has brought about difficulties in ensuring the reliability of these architectures. To overcome these challenges, innovative methods such as chaos engineering are becoming increasingly important.

Chaos engineering is a method of deliberately introducing faults or outages to different parts of the system to test the system's resilience and identify possible weak points. This method is especially indispensable for architectures with complex connections. This article highlights the importance of chaos engineering for microservices architectures and provides an overview of commonly used tools and technologies for chaos experiments. It also outlines best practices for planning and executing chaos experiments and discusses the benefits and challenges of using chaos engineering in practice. A project with a cluster along with microservices was used as a test environment to perform chaos experiments. Various chaos experiment scenarios have been introduced to the project, including Pod-based, Network-based, and Utilization-based experiments, utilizing open-source software such as Chaos Mesh and Litmus.

Keywords: Chaos Engineering, Cluster, Durability, Fault Tolerance, Kubernetes, Microservices, Pod, Production, Resilience, Simulation

TEŞEKKÜR

Bu araştırma için beni yönlendiren, karşılaştığım zorlukları bilgi ve tecrübesi ile aşmamda yardımcı olan değerli Danışman Hocam Doç. Dr. Mustafa Cem KASAPBAŞI'na teşekkürlerimi sunarım. Literatür araştırmalarımda yardımcı olan değerli hocalarım Prof. Dr. Rıfat YAZICI'ya ve Doç. Dr. Akhan AKBULUT'a teşekkür ederim.

Tezimin imalat aşamasındaki desteklerinde dolayı Bestcloudforme şirketine teşekkür ederim.

Tezimin her aşamasında beni yalnız bırakmayan aileme sonsuz sevgi ve saygılarımı sunarım.

Aybüke ERGÜL
İSTANBUL, 2024

1. GİRİŞ

Kaos mühendisliği, genellikle karmaşık yollarla etkileşime giren birçok farklı bileşenden oluşan mikroservis mimarileri için hayati öneme sahiptir. Bu mimariler, dayanıklılıklarını değerlendirmek ve potansiyel zayıf noktaları belirlemek için kaos deneylerine tabi tutulur. Netflix mühendisleri, üretim ortamlarını etkileyen ve önceden bilinmeyen davranışları keşfetmek için varsayımlar üzerine deneyler oluşturmak amacıyla Kaos Mühendisliği adlı bir yaklaşım geliştirdiler. Bu, 2010 yılında Kaos Maymunu adlı bir aracın tanıtılmasına yol açtı; bu araç, o tarihten bu yana hala aynı temel işlevselliği sunmaya devam etmektedir. Kaos Maymunu'nun amacı, üretimde rastgele bir çalışan örneği seçip onu sonlandırmaktır. Kaos Maymunu'ndan bu yana çeşitli Kaos Mühendisliği araçları ortaya çıkmıştır.

Hangi teknik veya araçların kullanılacağına karar verilirken; çalışmanın amacı, sistem durumu ve mevcut kaynaklar göz önünde bulundurulur. Bu çalışma, sistemdeki mevcut zayıf noktaları belirlemeyi ve genel sistem kararlılığını artırmayı amaçlar; bu kavram Al-Kuwaiti ve arkadaşlarının çalışmasında vurgulanmaktadır. Bu deneyler, ağ kesintileri, hizmetin kullanılamaz hale gelmesi veya kaynak tükenmesi gibi çoklu hata senaryolarını simüle eder ve sistemin bu senaryolara tepkisini gözlemler. Kaos mühendisliği, dağıtık hizmet mimarilerinde hata senaryolarını test etmek ve önceden bilinmeyen davranışları ortaya çıkarmak amacıyla ortaya çıkmıştır. Yük artışları, sistem hataları ve kesintiler müşteri memnuniyeti için önemlidir, bu nedenle sistem mimarisinin bu zorluklara nasıl yanıt vereceğini öngörmek gereklidir. Addeen'in dinamik hata toleransı modellerine yönelik araştırmasında yansıtıldığı gibi, kaos mühendisliğinin nihai amacı, sistemin olumsuz koşullar altında beklenen şekilde davrandığını doğrulamak ve eğer öyle değilse, dayanıklılığını artırmanın yollarını anlamaktır. Bir kaos deneyi yürütmek için öncelikle deneyin kapsamı tanımlanmalı, test edilecek mikroservisler ve bileşenler belirlenmeli ve belirli teknikler kullanılarak birden fazla hata senaryosu üretilmelidir. Ayrıca simüle edilecek hata senaryolarının yanı sıra deneyin sonuçlarını değerlendirmek için kullanılacak herhangi bir metrik veya başarı kriteri belirlenmelidir. Deney başladığında, hata senaryoları uygulanır ve sistemin tepkisi izlenir. Deney

sonrası, sonuçlar geliştirilmesi gereken alanları belirlemek amacıyla analiz edilir ve sistem gerektiği şekilde değiştirilir. Zorn'un mikroservis mimarilerinde dayanıklılık senaryoları üzerine yaptığı çalışmada vurgulandığı gibi, düzenli kaos deneyleri yaparak mikroservis mimarisinin dayanıklılığı ve güvenilirliği sağlanabilir. Bu makalenin temel amacı, mikroservis mimarilerinde kaos mühendisliğine kapsamlı bir genel bakış sağlamak ve bu mühendislikte kullanılan araçlarla daha dayanıklı ve güvenilir sistemler oluşturmanın yollarını araştırmaktır. Mikroservis mimarileri sürekli olarak evrildiğinden ve modern yazılımda önemli bir rol oynadığından, güvenilirliklerini korumak için sağlam stratejiler geliştirmek zorunlu hale gelmiştir. Bu çalışmanın merkezinde, ağ ve hizmet kesintileri, kaynak tükenmesi gibi çeşitli hata senaryoları mikroservis mimarileri için uygulanmakta ve sistem tepkileri incelenmektedir.

2. LİTERATÜR ÖZETİ

Bu bölüm, mevcut çalışma ile doğrudan ilgili olan temel kavramlar ve metodolojiler üzerine bir analiz sunmaktadır. Bölüm 2.1. Hata Enjeksiyonu, kaos mühendisliğinde önemli olan hata enjeksiyonu tekniklerini ve hataları incelemektedir. Bölüm 2.2. Senaryo Tabanlı Dayanıklılık Testi, mikroservis mimarilerinde direnç testi için çeşitli stratejilere odaklanmaktadır. Sistem dayanıklılığını değerlendirmek ve sistemdeki zayıflıkları belirlemek için gerçekçi senaryo tabanlı yöntemlerin önemini tartışmaktadır. Bölüm 2.3. Mikroservisler ve Bulut Mimarisinde Simülasyon, mikroservisler ve bulut mimarilerinde kullanılan simülasyon araçlarını ve tekniklerini incelemektedir. Ortamlardaki simülasyon, kaos deneylerinin gerçekleştirilmesi ve rafine edilmesi için kritik öneme sahiptir.

2.1. Hata Enjeksiyonu

Hata enjeksiyonu genellikle test ortamlarında, ortamın kontrol altında olduğu durumlarda uygulanır ve sistemin dayanıklılığını, sisteme kasıtlı olarak hatalar enjekte ederek ölçer. Bu yöntem, üretim ortamlarında ortaya çıkabilecek çeşitli hata senaryolarını öngörmek ve bu senaryoların sonuçlarını simüle etmek için yaygındır. Ana amacı, sistem mimarisindeki zayıflıkları ortaya çıkarmak ve belirsizliği en aza indirmektir. Burada, mevcut sistem gereksinimleri ile sağlanan hizmet arasındaki bağlantıyı anlamak önemlidir. Hata enjeksiyonu, sistemin performansını ve sistem gereksinimlerini beş önemli kavramla tanımlar; güvenilirlik, hata toleransı, güvenlik ve sürdürülebilirlik. Sistem içerisindeki bu 5 tanımda belirtilen hizmetlerin bu performans parametreleri ile nasıl ilişkilendirileceğini dikkate almak gereklidir. Örneğin, hata toleransı, bir sistemin veya bileşenin donanım veya yazılım hatalarına rağmen normal operasyonlarını sürdürebilme yeteneğini ifade eder. Hata toleransına sahip bir sistem, hataların varlığında programlarını ve giriş/çıkış işlevlerini doğru bir şekilde yürütmeye devam edebilen bir sistemdir.

Hata Enjeksiyonu tekniklerinin modern bir versiyonu olan Kaos Mühendisliği, Kaos Teorisi'nin matematiksel konseptinden etkilenmiştir. Bu yöntem,

geleneksel test yerine üretim ortamlarına entegre edilen bir tekniktir ve sistemlerin öngörülen hatalarının yanı sıra öngörülemeyen hatalarını da ele alır. Üretim ortamlarındaki öngörülemezliği kabul eder ve bu öngörülemezlik ile çalışır; sistemlerin baskı altında nasıl tepki vereceğine dair yaklaşımlar sunar. Daha dayanıklı ve sağlam sistemler geliştirilmesine yardımcı olur. Bu sistemlerin beklenmedik koşullara uyum sağlaması ve hayatta kalması hedeflenir.

Hata enjeksiyonu veya Kaos Mühendisliği, yalnızca sistem için bir test süreci değil, aynı zamanda sistem geliştirme ve bakım yaşam döngüsünün temel bir parçasıdır. Sistem mühendislerine, sistemlerin üretim ortamlarında nasıl davranabileceğine dair perspektifler sunar. Bu şekilde, yalnızca güvenilir ve etkili değil, aynı zamanda kritik altyapı ve yüksek öneme sahip hizmet alanlarında karşılaştıkları sürekli değişen zorluklarla başa çıkacak kadar esnek sistemler yaratmada önemlidir.

2.2. Senaryo Tabanlı Dayanıklılık Testi

Senaryo tabanlı dayanıklılık testi, bir sistemin belirli koşullar ve senaryolar altında nasıl performans gösterdiğini ve dayanıklılığını değerlendirmeye dayanır. Sistem davranışını inceler ve kalitesini tanımlar. Bu ortamlar ve senaryolar genellikle gelişmiş izleme sonuçlarına veya sistem sahiplerinin uzmanlığına dayalı olarak tanımlanır. Senaryolar önceden belirlenir ve sistem özellikleri (performans, güvenilirlik, kullanılabilirlik vb.) bu senaryolar içinde incelenir. Bu, sistemin karşılaşabileceği gelecekteki olası değişiklikleri ve zorlukları temsil eder ve böylece sistemin uzun vadeli geçerliliği ve dayanıklılığı hakkında bilgiler sağlar. Sistemden rutin operasyonlar sırasında beklenen performans hakkında net bir çıkarım sunar.

Bu durumda, senaryoları belirlemek ve araştırmak için sistem sınırlarını öğrenmek gereklidir. Sistem sınırları, performans, güvenlik ve güvenilirlik parametreleri göz önünde bulundurularak araştırılır. Bir sistemin olağandışı koşullar altında nasıl davrandığını anlamak ve kapsamlı bir dayanıklılık değerlendirmesi sağlamak için senaryolar belirlenmeli ve senaryo tabanlı testler gerçekleştirilmelidir.

Bu senaryoları belirlemek için kullanılan bazı çalışmalar vardır. Risk Tabanlı Hata Enjeksiyonu (RDFI), bir sistemde en yüksek risk taşıyan alanları belirler. Bu riskler belirlendikten sonra, Kaos Mühendisliği senaryoları bu alanlara odaklanır. Özellikle siber güvenlik alanında, RDFI gibi yenilikçi yaklaşımlar hangi senaryoların daha kritik ve değerli olduğunu belirler. Bu şekilde, sistemlere gerçekçi ancak kontrollü hatalar enjekte edilerek sistemin zayıflıkları ortaya çıkarılabilir ve sistemlerin siber saldırılara karşı genel dayanıklılığı artırılabilir. RDFI, kaos mühendisliği prensiplerini uygulayan bir bulut güvenlik sistemi olan CloudStrike dahil olmak üzere çeşitli sistemlerde uygulanabilen bir tekniktir. RDFI'ye ek olarak, senaryoların analizi için kullanılan ve kaos mühendisliği ile uyumlu diğer teknolojiler de vardır. Mimari Tercih Analizi Yöntemi (ATAM) gibi araçlar kullanılarak verimli bir şekilde yapılandırılabilir ve düzenlenebilir. ATAM, belirli senaryolar kullanarak bir sistemin mimari kararlarını değerlendirir. Bu senaryolar, sistemin karşılaşabileceği çeşitli durumları temsil eder ve sistemin bu durumlarda nasıl performans göstereceğini test eder. Kısacası, sistemin mimari tasarım kararlarını analiz etmek için kullanılan bir yöntemdir. Bu yöntem, senaryoların avantajlarını, dezavantajlarını ve risklerini ortaya koyar.

Senaryo tabanlı dayanıklılık testleri, bir sistemin dayanıklılığı hakkında birçok perspektif sağlar. Sistem geliştiricilerine, öngörülemeyen durumlar karşısında sağlam sistemler tasarlama konusunda rehberlik eder. Bu yaklaşım, dinamik ve zorlu operasyonel ortamlarda sistemlerin güvenilirliğini ve kararlılığını sağlamak açısından kritik öneme sahiptir.

Mikroservis mimarilerinin güvenliği ve dayanıklılığı açısından, Kaos Mühendisliği deneylerinin senaryolarına öncelik vermek sistematik bir yaklaşım gerektirir. Bu bağlamda, deneyler Jernberg ve arkadaşlarının önerdiği gibi üç ana gruba ayrılabilir: Birinci grup, hedef uygulamaya uygulanamayan deneyleri içerir; ikinci grup, uygulamanın ideal davranış sergilememesi ve izleme araçlarının eksikliğiyle sonuçlanabilecek deneylerden oluşur; üçüncü grup ise uygulamanın iyi performans gösterdiği deneyleri içerir. Deneylere öncelik vermenin bir diğer yöntemi, Basiri ve arkadaşlarının önerdiği iki kritere dayanır: (1) etkilenen sistemin kritikliği ne kadar yüksekse, öncelik de o kadar yüksek

olmalıdır ve (2) başarısızlıklara neden olan enjeksiyonlar, yürütmeyi yavaşlatan veya gecikmeler ekleyen enjeksiyonlardan daha yüksek önceliğe sahip olmalıdır.

2.3. Mikroservisler ve Bulut Mimarisinde Simülasyon

Kaos deneyleri, mikroservisler ve bulut mimarilerinin dayanıklılığını ölçmek için çok önemlidir. Bu deneyler, bu sistemlerin beklenmedik durumlara tepkisini ölçer. Ancak, bu tür çalışmaları doğrudan üretim ortamlarında yürütmek, sisteme uygulanacak hatalar nedeniyle risklidir ve son kullanıcı deneyimini olumsuz etkileyebilir. Bu riski azaltmak için simülasyon tabanlı yaklaşımlar kullanılabilir. Simülasyon tabanlı yaklaşımlar, üretim ortamına tıpatıp benzeyen ortamları simüle ederek hata enjeksiyonlarının son kullanıcıları etkilemesini önler. Bu deneylerin güvenli bir şekilde gerçekleştirilmesini sağlar.

Mikroservisler ve bulut mimarileri üzerinde stres testleri yapmak için tasarlanmış çeşitli simülasyon araçları vardır. Bu araçlar arasında Misim, SimuLizar, Slingshot, Mqsim, MuSim, DRACeo, BigHouse, PacketStorm, iFogSim ve GreenCloud bulunmaktadır. Her biri, mikroservisler ve bulut mimarilerinin stres altındaki davranışlarını simüle etmek için özel olarak geliştirilmiştir. Örneğin, Misim aracı, genel dayanıklılık mekanizmaları ve çeşitli hata senaryolarını simüle etme yeteneği ile öne çıkar]. Bu simülatör, devre kesiciler, bağlantı sınırlayıcılar, yeniden denemeler, yük dengeleyiciler ve otomatik ölçekleyiciler gibi dayanıklılık mekanizmalarını içerecek şekilde tasarlanmıştır. Ayrıca, örnek/hizmet sonlandırma ve gecikme enjeksiyonu gibi hata türlerini de simüle edebilir.

Bu simülasyon araçları, karmaşık mikroservis mimarileri ve bulut tabanlı sistemlerin çeşitli hata ve yük senaryoları altında nasıl performans gösterdiğini anlamak için değerli bir kaynaktır. Özellikle, bu araçlar sayesinde sistemin zamansal davranışları ve dayanıklılık mekanizmalarının etkinliği hakkında zengin nicel veriler toplanabilir. Bu simülasyonlar, TeaStore gibi referans mimariler üzerinde gerçekleştirildiğinde, sistemlerin ölçeklenebilirlik ve hata toleransı yetenekleri hakkında ayrıntılı bilgiler sağlar.

Canlı üretim ortamlarındaki riskleri daha da azaltmak için, Kubernetes kümeleri gibi platformlarda çalışan sistemler için ön üretim ortamlarının kullanılması

önerilir. Bu tür ortamlar, canlı sistemlerin neredeyse birebir kopyaları olabilir ve gerçek dünya senaryolarını simüle etmek için mükemmel test alanları sağlar. Ön üretim ortamlarında test yapmak, sistemlerin çeşitli senaryolar altındaki davranışlarını gözlemleme fırsatı sunar ve bu yaklaşım, özellikle mikroservis mimarilerinin ve bulut tabanlı sistemlerin karmaşık doğasını dikkate alarak, ayrıntılı ve kapsamlı bir dayanıklılık değerlendirmesi yapılmasını sağlar. Bu strateji, sistemlerin canlı bir ortama alınmadan önce her türlü stres ve hata durumuna karşı hazırlıklı olmasını sağlar ve böylece sistemlerin beklenmedik olaylar karşısında dayanıklı ve esnek kalmasını garanti eder.

3. MATERYAL VE METODOLOJİ

Bu bölüm, mikroservis mimarilerinde güvenliği ve dayanıklılığı artırmak için kaos mühendisliği deneylerinin metodolojik çerçevesini detaylandırmaktadır. Araştırmanın yapı taşları; araç seçimi, sistem yapılandırması, deneysel tasarım ve uygulama, izleme ve analiz metodolojileri, değerlendirme metrikleri ve karşılaşılan zorluklar yoluyla elde edilen derinlemesine içgörüleri kapsar.

Litmus, bulut yerel altyapılarda ve uygulamalarda kaos testleri gerçekleştiren bir kaos mühendisliği platformudur. Bu nedenle, Kubernetes'te pod'ları sonlandırmak, Kafka kümelerinde düşük seviyeli hataları simüle etmek veya AWS örneklerini sonlandırmak gibi işlemleri gerçekleştirebilir. Chaos Mesh, Kubernetes ortamları için bulut yerel bir kaos mühendisliği platformudur. Kernel paniği, kaynak tüketimi, gecikme, bağlantı kaybı ve depolama gecikmesi gibi geniş bir yelpazede saldırıları destekler. Chaos Mesh ve Litmus Chaos gibi önde gelen kaos mühendisliği araçları bu deneylerin uygulanması sırasında kullanılmış ve böylece sistemin farklı hata koşulları altında nasıl performans gösterdiği kapsamlı bir şekilde analiz edilmiştir. Kaos deneylerinin etkileri, Prometheus ve Grafana izleme araçları kullanılarak deney sonuçlarını izlemek ve analiz etmek suretiyle gerçek zamanlı olarak gözlemlenmiş ve değerlendirilmiştir. Bu bölümde, araştırma sürecinin temel adımları, kullanılan araç ve tekniklerin seçimi, gerçekleştirilen deneylerin kapsamı ve deneylerden elde edilen bulgular kapsamlı bir şekilde sunulmaktadır.

3.1. Kaos Mühendisliği Araçlarının Seçimi

Her organizasyonun, altyapısına ve ihtiyaçlarına en uygun araçları seçmesi esastır. Bu çalışmada incelenen proje, dört mikroservisten oluşan bir yapıda yürütülmektedir. Proje; dev, ön üretim ve üretim olmak üzere üç farklı Kubernetes kümesi üzerinde yer almaktadır. Kaos mühendisliği deneyleri, gerçek üretim ortamının koşullarını en iyi şekilde yansıtan ve en gerçekçi sonuçları elde etmemizi sağlayacak ön üretim ortamında gerçekleştirilmiştir. Her ortam, yerel bir altyapıda RKE2 Kubernetes kümesi üzerinde çalışmaktadır.

Kaos mühendisliği deneyleri için Chaos Mesh ve Litmus Chaos araçları tercih edilmiştir. Her iki aracın da seçilme nedenleri, Kubernetes tabanlı mikroservis mimarilerine özel olmaları ve açık kaynak olmalarıdır. Bu özellikler, projenin çeşitli ihtiyaçlarını karşılayacak esnekliği ve geniş bir senaryo yelpazesini içermektedir.

Litmus Chaos, geniş kaos deney kütüphanesi ile öne çıkarken, kullanıcıların çeşitli senaryoları hızla denemelerine olanak tanır; Chaos Mesh ise kullanıcı dostu web arayüzü sayesinde karmaşık kaos deneylerinin kolayca tasarlanıp uygulanmasını sağlar. Bu iki aracın birlikte kullanılması, deneylerin çeşitliliğini ve derinliğini artırarak, sistemimizin çeşitli hata senaryolarına karşı dayanıklılığını kapsamlı bir şekilde değerlendirme fırsatı sunar.

Bu araçlar, CPU doygunluğu, disk doluluğu, yavaş HTTP istekleri ve rastgele pod silme gibi geniş bir deney senaryosu yelpazesi sunar. Litmus Chaos ve Chaos Mesh'in birlikte kullanılması, deneylerin çeşitliliğini artırarak, sistemimizin farklı hata senaryolarına karşı dayanıklılığını kapsamlı bir şekilde değerlendirme imkanı sağlar. Ayrıca, özelleştirilebilir senaryolar ve detaylı analiz fırsatları sunarak zayıf noktaların tespit edilmesi ve süreçlerin iyileştirilmesi açısından önemli bir avantaj sağlarlar.

Bu seçim, sistemimizin karmaşıklığını ve mikroservis mimarilerinin özelliklerini kapsamlı bir şekilde ele almayı amaçlamaktadır.

3.2. Küme ve Mikroservisler Genel Bakışı

Proje, mikroservis tabanlı uygulamaları geliştirmek, test etmek ve üretime almak için üç farklı Kubernetes küme ortamına (geliştirme, ön üretim ve üretim) sahiptir. Bu çalışma, ön üretim ortamında yapılmıştır. Her küme, yerel bir altyapıda Rancher Kubernetes Engine 2 (RKE2) ile yapılandırılmıştır ve her küme, maksimum erişilebilirlik ve sistem güvenilirliği sağlamak amacıyla yüksek erişilebilirlik ilkelerine göre ayarlanmış 3 master ve 3 worker düğümünden oluşmaktadır. Üç ortam da benzer mimari ile çalışmaktadır. Topolojilerinde bir fark yoktur. Çalışmanın üretim ortamında yapılması riskli olacağından, ön üretim ortamı tercih edilmiştir. Burada önemli olan, ön üretim ortamının üretim ortamı ile aynı olmasıdır.

Araştırmanın güvenilirliğini artırmak için kullanılan RKE2 sürümü aşağıda belirtilmiştir:

rke2 version v1.26.11+rke2r1
go version go1.20.11

Hizmetin yönetilebilirliğini ve ölçeklenebilirliğini artırmak için mikroservis mimarisi benimsendi. Kimlik doğrulama, backend, frontend ve mail API gibi dört temel mikroservis, belirli işlevleri yerine getirerek uygulamayı oluşturur. Her mikroservis aynı küme üzerinde çalışır ve birbiriyle iletişim kurar.

Python ve FastAPI framework'ü kullanılarak geliştirilen Backend mikroservisi, uygulamanın operasyonel omurgası olarak hizmet eder. Diğer üç hizmetle etkileşimler backend üzerinden gerçekleştirilir, temel işlevler yönetilir, doğrudan MySQL veritabanı bağlantıları yönetilir ve sorgular bu mikroservis aracılığıyla yürütülür. Sorguları işleme ve kullanıcılara yanıtları geri gönderme konusunda büyük bir rol oynar, uygulama mantığını ve veri yönetimini merkezileştirir. Frontend mikroservisi, React ile geliştirilen ve Nginx aracılığıyla sunulan kullanıcı arayüzünü sağlar. Backend hizmetinden alınan yanıtları kullanıcıya iletir. Kullanıcının ilk etkileşime geçtiği ve gördüğü yer doğrudan frontend mikroservisi ile ilgilidir. Burada ağ gecikmesi ve paket kaybı gibi durumlar doğrudan kullanıcıyı etkiler. Mail API mikroservisi, özellikle e-posta yönetimi ve gönderimi için Amazon Web Services (AWS) ile entegre edilmiştir ve Simple Email Service (SES) kullanılarak çalışır. Bu hizmet, kullanıcılara kimlik doğrulama ve şifre sıfırlama işlemleri için e-posta göndererek sistemin iletişim çerçevesini oluşturur. Kimlik Doğrulama mikroservisi, uygulama içinde güvenli erişim ve etkileşimi sağlamak için önemlidir. Kullanıcılara ait tokenların oluşturulmasından ve doğrulanmasından sorumludur. Bu tokenları saklamak için Redis Sentinel ile doğrudan iletişim kurar. Backend hizmeti, bu tokenları kimlik doğrulama hizmetine doğrudan taleplerle doğrular ve doğrulanmış kullanıcı işlemlerini sağlar. Ayrıca, mikroservisleri destekleyen MySQL ve Redis Sentinel, ilişkisel veri depolama ve yüksek kullanılabilirlikli önbellekleme çözümleri sunar. Bu çalışmada, kaos mühendisliği senaryolarımız için uygulamanın mikroservis mimarisine odaklanıyoruz. Veritabanı katmanlarına

uygulanabilecek potansiyel kaos senaryoları olmasına rağmen, bu tür incelemeler mevcut araştırmamızın kapsamı dışında kalmaktadır.

3.3. Kaos Mühendisliği Uygulaması

Bu bölüm, mikroservis mimarilerinin kaos mühendisliği deneylerinin uygulama sürecini kapsamaktadır. Ortamı tanıdıktan sonra, ortama uygun senaryoları belirleyeceğimiz ve bu senaryoları sisteme entegre edeceğimiz bir bölüm olacaktır. Bu süreçte, senaryoları belirlerken nelere dikkat ettiğimizi ve senaryoları nasıl entegre ettiğimizi de ele alacağız.

Kaos mühendisliği deneylerini başarıyla uygulayabilmek için öncelikle sistemin kritik bileşenleri ve hizmetleri belirlenmelidir. Bu tespit aşaması, sistem mimarisi ve operasyonel dinamikler hakkında detaylı bir anlayış gerektirir. Sistemin mimarisini ve işleyişini bilmek önemlidir. Bu bileşenler ve hizmetler için oluşturulacak hata senaryoları, sistemimizin hangi koşullar altında kırılgan olduğunu ve hangi koşullar altında yüksek dayanıklılık gösterdiğini anlamamıza yardımcı olacaktır. Deneylerin uygulanabilmesi için önceden belirlenmiş Chaos Mesh ve Litmus Chaos araçları kullanılarak çeşitli hata senaryoları gerçekleştirilecektir.

3.3.1. Senaryo Tasarımı ve Seçimi

Senaryoları belirlemek için sistemi iyi tanımak ve hangi senaryoların sistem için doğru senaryolar olduğunu belirlemek gereklidir. Ayrıca, senaryoları belirlerken olası gerçek sorunlara yakınlık önemlidir. Burada istenen, olası bir gerçek senaryoda sistemin tepkisini gözlemleyerek önceden önlem almaktır. Her hata senaryosu, üretim ortamında gerçekleşebilecek bir olayı gerçekçi bir şekilde temsil etmelidir. Bu nedenle, senaryoları belirlerken dikkate alınması gereken birçok parametre vardır. Sistemimizin mimarisi, mikroservislerin birbirine bağımlılığı ve her hizmetin genel uygulama performansındaki kritikliği, senaryo seçimine yaklaşımımızı belirler. Bu parametreleri dikkate alarak, seçilen kaos mühendisliği senaryoları şunlardır:

Birinci Senaryo: Pod Hatası

Pod'lar, Kubernetes'teki en küçük birimlerdir. Her pod, uygulamanın bir bölümünü çalıştırır. Bu bölümde meydana gelebilecek herhangi bir sorun tüm sistemi bozabilir. Bu nedenle Pod Hatası senaryosu önemlidir; Gerçekçi bir şekilde ortaya çıkabilecek pod düzeyindeki sorunlara karşı hizmetlerimizin dayanıklılığını test eder. Pod'ların geçici olduğu ve herhangi bir zamanda yeniden planlanabileceği, sonlandırılabileceği veya yeniden başlatılabileceği göz önüne alındığında, sistemin pod kesintileriyle nasıl başa çıktığını anlamak, yüksek erişilebilirlik ve dayanıklılığı sağlamak için kritiktir. Bu, hizmetlerimizin gerçekçi bir şekilde ortaya çıkabilecek pod düzeyindeki sorunlara karşı dayanıklılığını test eder.

Mimarımdaki en kritik mikroservis Backend hizmetidir. Bu hizmet, sadece operasyonel omurga olmakla kalmayıp, diğer mikroservislerle yoğun bir şekilde iletişim kurar, veritabanı etkileşimlerini yönetir ve sorguları işler, aynı zamanda uygulama mantığının ve veri yönetiminin merkezi bir noktasıdır. Karmaşık yapısı ve geniş iletişim arayüzleri, onu kesintilere karşı özellikle savunmasız hale getirir. Bu nedenle, pod hatası deneylerimizin odak noktası olarak seçilmiştir.

Pod Hatası Deneylerinin Uygulanması: Bu aşamada iki senaryoyu ele alacağız. Bunlar Pod Arızası ve Pod Sonlandırma senaryolarıdır. Bu deneyler, düğüm ölümü, sistem kesintileri veya yazılım hataları gibi çeşitli hatalar nedeniyle pod'ların beklenmedik şekilde sonlandırılmasını simüle etmeyi amaçlamaktadır. Pod arızası senaryosunda, belirli pod'lara rastgele arızalar verilir. Rastgele bir arıza karşısında hizmetin ve sistemin dayanıklılığı test edilir. Burada, backend hizmeti pod'larının rastgele arızasını ve bunun kalan mikroservisleri nasıl etkileyeceğini gözlemleyeceğiz. Pod sonlandırma senaryosu, belirli pod'ların ani bir şekilde sonlandırılmasını içerir. Buradaki amaç, sistemin ani kayıplara tepkisini öğrenmektir.

İkinci Senaryo: Ağ Saldırısı

Ağ saldırısı senaryosu, sistemin ağla ilgili sorunlara veya saldırılara nasıl tepki vereceğini test etmek için gerçekleştirilir. Ağ gecikmesi, paket kaybı, ağ parçalanması ve düşük bant genişliği gibi durumları simüle eder. DDoS

(Dağıtılmış Hizmet Reddi) saldırısı da ağ senaryoları ile simüle edilebilir. Bu senaryolar, mikroservisler arasındaki iletişim göz önünde bulundurulduğunda kritiktir. Bu simülasyonlar, sistemin dayanıklılığını, zayıf noktalarını ve hizmet sürekliliğini test eder. Sistemimizin bu tür saldırılar karşısında bütünlüğünü koruması önemlidir.

Üçüncü Senaryo: Stres Testi

Stres testleri, sistemi sınırlarına kadar zorlayarak, sistemin maksimum kapasitesinde nerede zorlandığını, nasıl tepki verdiğini ve hangi koşullar altında başarısız olabileceğini gözlemler, normal çalışma koşullarından ziyade bu durumları inceler. Bu başlık altında, yüksek CPU yükü, bellek kısıtlamaları veya yoğun I/O işlemleri gibi senaryolar oluşturulabilir. Sistemimizin ölçeklenebilirliğini ve esnekliğini değerlendirmek kritiktir. Mikroservislerimizin stres altında nasıl davrandığını izleyerek, beklenmedik kapasite sorunlarını tespit edebilir ve sistemdeki zayıf noktaları önceden belirleyebiliriz.

Senaryoların belirlenmesi sürecinde, seçilen her deney için mikroservislerimizin akışını etkileyebileceğini düşündüğümüz ana sorunlara öncelik verildi. Deneyler test ortamında gerçekleştirildiğinden, üretim ortamıyla uyum sağlanmıştır. Bu senaryolar sadece sistem için testler değil, aynı zamanda sağlam bir platformu sürdürülebilir kılmak için de önemlidir. Her senaryoda, zayıf noktaları ortaya çıkarmayı ve hizmetlerimizin güvenilirliğini artırmak için proaktif çözümler tasarlamayı amaçlıyoruz.

3.3.2. Kaos Senaryolarının Yürütülmesi

Bu bölümde, sistemimiz için belirlediğimiz senaryoları uygulayacağız. Kaos senaryoları ile mikroservislerin beklenmedik hatalara nasıl tepki verdiğini gözlemleyeceğiz. Sistem dayanıklılığını anlamak için metriklerin ve sürecin dikkatlice izlenmesi gerekmektedir. Bu senaryoları uygularken süreç kontrollü bir ortamda gerçekleştirildi.

Birinci Senaryo: Pod Hatası

Kaos deneylerine başlamadan önce, Kubernetes kümesine erişim sağlamak için gerekli olan token ile yetkilendirme yapıldı. Bu, Chaos Mesh ve Litmus araçlarının küme üzerinde gerekli işlemleri gerçekleştirebilmesi için zorunludur. Doğrulama tamamlandıktan sonra, arayüzdeki deneyler bölümünde "pod hatası" seçilerek deney aşamasına başlandı. Deneyin gerçekleştirileceği namespace seçildi. Belirli bir backend pod'u için etiket seçildi. Ayarlamalar tamamlandıktan sonra, arayüz üzerinden "submit" yapılarak deney başlatıldı. Bu adımlar hem Pod Hatası hem de Pod Sonlandırma senaryoları için uygulandı. Deney başladıktan sonra, pod'ların durumu küme terminali üzerinden anlık olarak izlendi.

İkinci Senaryo: Ağ Saldırısı

İkinci senaryoda, mikroservis mimarilerinin dayanıklılığını değerlendirmek amacıyla bir ağ saldırısı simülasyonu gerçekleştirildi. Bu test sırasında, dağıtılmış hizmet reddi (DDoS) saldırısını taklit etmek için ağ iletişimlerinde yapay gecikmeler oluşturuldu. Ön uçta belirgin yavaşlamalar oldu; yanıt süreleri neredeyse %300 artarak ortalama 100 milisaniyeden 300 milisaniyenin üzerine çıktı. Bu durum, kullanıcıların uzun bekleme süreleri ve aralıklı zaman aşımı sorunları yaşaması nedeniyle kullanıcı deneyimini doğrudan etkiledi.

Üçüncü Senaryo: Stres Testi

Stres senaryosunda, sistemin kapasitesi artırıldıkça izleme yapıldı. CPU ve bellek kullanımı kasıtlı olarak kapasitenin %90'ına çıkarıldı. Sürekli olarak MySQL veritabanı ile etkileşimde bulunan arka uç mikroservisi zorlanma belirtileri gösterdi. Bu hizmet normalde ortalama 100 milisaniyede yanıt dönerken, bu süre 800 milisaniyenin üzerine çıkarak kullanıcı deneyimini olumsuz etkiledi. İkinci bir gözlem olarak, veritabanı sorguları veya eklemeleri yapılırken sistem hatalarla karşılaştı, bu da kullanıcı isteklerinin zaman aşımına uğramasına neden oldu.

Stres senaryosu sırasında performansta belirgin bir düşüş gözlemlendi. Bu alana daha fazla odaklanmamız gerektiğini ve iyileştirilmesi gereken alanlar olduğunu

öğrendik. Önceden tanımlanmış otomatik ölçeklendirme ayarlarımız, yükteki ani artış için yeterli olmadı; bu da gerçek zamanlı talebe göre kaynakları dinamik olarak ayarlayabilen daha duyarlı ölçeklendirme mekanizmalarına ihtiyaç duyulduğunu gösteriyor. Ayrıca, yüksek yük koşullarında veritabanı bağlantıları veya istek işleme yavaş göründü; bu da veritabanı yönetimi ve sorgu stratejilerimizde potansiyel iyileştirmeler yapılması gerektiğine işaret ediyor.

3.4. İzleme ve Analiz

Senaryolar uygulandıktan sonra sistemin davranışı izlenmiş ve değerlendirilmiştir. Hata enjeksiyonu sürecinde sistemin sürekli olarak izlenmesi ve analiz edilmesi, hataların etkisini anlamak için çok önemlidir. Senaryolar sırasında, sistem performansı, kaynak kullanımı ve hizmet durumu hakkında gerçek zamanlı veriler toplanmaya çalışıldı. Kaos deneylerinin mikroservis mimarisi üzerindeki etkileri incelendi. Bu süreçte belirli izleme araçları kurulabilir veya araçlarında arayüzün kullanılarak gerçek zamanlı veriler toplanabilir. Bu aşamada metrikleri toplayabilmek için Prometheus ve Grafana kurulumları yapıldı. Küme içine kurulan Grafana ve Prometheus sayesinde; sistem performansı, kaynak kullanımı ve gerçek zamanlı işlemler takip edildi. Bu araçlar izlemek için birincil araç olarak kullanıldı. Grafana, kümedeki metrikleri görselleştirmek için güçlü ve anlaşılır bir arayüz sundu. Bu nedenle, sistemimizin sağlık durumunu ve performansını sürekli izlemek için Grafana'yı tercih ettik. Ardından Grafana'nın Prometheus ile entegrasyonu tamamlandı ve bilgiler Promethues tarafından aktarıldı.

Grafana'ya ek olarak, kaos deneylerini izlemek ve yönetmek için Chaos Mesh ve Litmus arayüzü de kullanıldı. Bu arayüz, kaos deneylerinin hangi aşamada olduğunu anlamamıza olanak tanıdı, böylece deneylerin etkilerini daha iyi takip edebildik. Ayrıca, deneyler sırasında küme terminali aktif olarak kullanıldı. Küme terminali üzerinde pod'ların mevcut durumu izlenmiş ve 'describe' komutu ile CPU/bellek gibi önemli metriklere bakılarak deneyler gerçekleştirilmiştir.

4. ARAŞTIRMA BULGULARI VE TARTIŞMA

Kaos mühendisliği uygulamalarımızda gerçekleştirdiğimiz ilk senaryo "pod hatası" idi. Bu senaryoda ana odak, rastgele veya belirli bir hizmetin kesintiye uğraması veya hata durumunda sistemimizin nasıl tepki vereceğini gözlemlemekti. Pod hatası senaryosu sırasında, hata sürecinde backend hizmetimizin hiçbir şekilde yanıt vermediğini gözlemledik. Bu durum, kullanıcıların kullanıcı arayüzünde (UI) herhangi bir işlev gerçekleştirememesine neden oldu. Sistemin toparlanması yaklaşık 180 saniye sürdü. Sistem üzerindeki böyle bir kesinti, kullanıcı deneyimini olumsuz etkiler. Sistemin bir hata durumunda 3 dakika içinde toparlanamaması, bu tür hatalara karşı önlem almanın gerekliliğini vurgulamaktadır. Ancak "pod kill" senaryosunda, bir backend pod'u arızalandığında, başka bir pod'un hızla devreye girdiğini ve bu durumun kullanıcıya hiç yansımadığını gözlemledik. Küme içinde bir pod öldüğünde, talepleri başka bir pod'a yönlendirmek için hızla yeni bir pod oluşturuldu. Sistem bu senaryoyu başarıyla geçti.

İkinci senaryomuzda, sistemin ağla ilgili stres altında nasıl tepki vereceğini test etmek amacıyla bir ağ saldırısı simülasyonu gerçekleştirildi. Chaos Mesh ve Litmus aracılığıyla yapay ağ gecikmesi ve paket kaybı ekleyerek dağıtılmış

hizmet reddi (DDoS) saldırısının etkilerini taklit etmeyi amaçladık. Grafana ile yapılan izlemelerde, normal koşullarda ortalama 200 milisaniye civarında olan ön uç hizmeti yanıt sürelerinin saldırı sırasında önemli ölçüde arttığı görüldü. Chaos Mesh ile yanıt süresi yaklaşık 600 milisaniyeye (yüzde 200 artış) çıkarken, Litmus ile 580 milisaniye (yüzde 190 artış) olarak kaydedildi. Paket kaybı oranı yüzde 5 olarak ayarlandı ve bu da mikroservisler arasındaki iletişim gecikmelerini daha da artırdı. Her iki araç da sistem performansı üzerinde benzer etkiler kaydetti, ancak Chaos Mesh ile gecikme artışları biraz daha yüksek oldu. Gecikme artışları, sistemin işlem kapasitesini doğrudan etkiledi ve saniyedeki istek sayısını (RPS) Chaos Mesh ile 1.000'den 500'e (yüzde 50 azalma) ve Litmus ile 1.000'den 520'ye (yüzde 48 azalma) düşürdü. Bu çıktılar sayesinde ağ altyapımızdaki zayıf noktaları gözlemledik ve bu tür riskleri azaltmak için geliştirilmiş ağ güvenliği önlemleri ve trafik yönetimi stratejilerine odaklanmamız gerektiğini öğrendik.

Üçüncü senaryo ise, sistemin ölçeklenebilirliğini ve kaynak yönetimini test etmek amacıyla kümenin CPU ve bellek kullanımını kasıtlı olarak maksimuma çıkardığımız bir stres testiydi. Bu test sırasında, backend mikroservisinin ortalama işlem süresi önemli ölçüde arttı ve genellikle istek başına yaklaşık 50 milisaniye yükseldi. Chaos Mesh ile işlem süreleri istek başına 250 milisaniyeye (yüzde 400 artış) çıkarken, Litmus ile 240 milisaniyeye (yüzde 380 artış) ulaştı. Sistem CPU kullanımı yüzde 90'a ulaştı ve bu da kümenin genel işlem verimliliğinde önemli bir düşüşe neden oldu. Bunun sonucunda, işlem kapasitesi Chaos Mesh ile 2.000 RPS'den 1.200 RPS'ye (yüzde 40 azalma) ve Litmus ile 1.250 RPS'ye (yüzde 37,5 azalma) düştü.

CPU ve bellek kaynakları doyuma ulaştığında, küme yavaşlamaya başladı, özellikle en fazla kaynak tüketen pod'larda bu durum belirginleşti. Bu, gecikmelerin artmasına ve uygulama performansında belirgin bir yavaşlamaya yol açtı. Bu sonuçlar, yüksek yük koşulları altında sistemin sınırlamalarını vurguladı ve kaynak tahsisini optimize etme, pod kaynak taleplerini ve sınırlarını ayarlama ve sistemin önemli performans kaybı olmadan zirve talep

dönemlerine yanıt verebilmesini sağlamak için daha agresif otomatik ölçeklendirme stratejileri uygulama gerekliliğini ortaya koydu.

Tablo 1. Araçlar / Senaryolar

Senaryo 1	Senaryo 2	Senaryo 3
85%	90%	80%
90%	85%	85%

Yukarıdaki tablo, Chaos Mesh ve Litmus araçlarını kullanarak yapılan kaos deneylerinde gözlemlenen önemli etkilerin nicel bir temsilidir. Bu sonuçlar, araçların farklı operasyonel stres senaryoları altında mikroservis mimarimiz üzerinde yaratabileceği etkileri göstermektedir. Her iki araç da benzer türde kaos deneyleri için kullanılmış ve benzer test senaryolarında benzer düzeyde etkililik göstermiştir. İlk yüzdeler Litmus aracına, ikinci değerler ise Chaos Mesh aracına aittir.

"Pod Hatası" senaryosunda, hem Chaos Mesh hem de Litmus, sistemin büyük bir kısmının devre dışı kalmasına neden olmuş ve sistemimizin pod hatalarını, genel hizmet sürekliliğini etkilemeden yönetme yeteneğinde kritik bir zayıflık olduğunu ortaya koymuştur. "Ağ Saldırısı" senaryosunda, sistem performansında başlangıçta beklenenden çok daha büyük bir bozulma fark edilmiştir. Chaos Mesh ağ gecikmesinde önemli bir artışa neden olurken, Litmus yanıt sürelerini daha da kötüleştirerek kullanıcı deneyimini olumsuz etkileyen biraz daha şiddetli etkiler sergilemiştir. CPU ve bellek kaynaklarının sınırlarına kadar zorlandığı "Stres Testi" senaryosunda, her iki araç da sistemin kararlılığı ve performansı üzerinde önemli ölçüde etkili olmuştur. Chaos Mesh, sistemde belirgin bir bozulmaya neden olurken, Litmus da aşırı yük koşulları altında sistem performansı üzerinde aynı etkiye sahip olmuştur.

5. SONUÇ VE ÖNERİLER

En önemli zorluklardan biri, test ortamı ile üretim ortamlarının genellikle benzer olmamasıdır. Test ortamı, üretim ortamı ile aynı olacak şekilde tasarlanmış olsa dahi üretim ortamındaki karmaşıklığı bulmak zor olacaktır. Fakat üretim ortamında yapılacak deneyler ise, müşteri memnuniyetini olumsuz etkileyebilir çünkü sistemin nasıl bir cevap vereceği belirsizdir. Üretime özgü bazı davranışlar ve sorunlar üretimden önce ortaya çıkmayabileceğinden, bu durum kaos deneylerinin etkinliğini azaltabilir. Bu nedenle, deneylere başlamadan önce, test ortamı ile üretim ortamının gerçekleştirilecek senaryolar açısından senkronize olduğunu gözlemlemek gereklidir. Deneylerin doğruluğunu sağlamak için, test ortamlarındaki ayarların ve yapılandırmaların üretim ortamını yansıtması gerekmektedir. Bu uyum, üretim ortamının kaos olaylarına vereceği gerçek tepkileri ölçmek için önemlidir.

Bir diğer önemli nokta, senaryoların mikroservisler üzerindeki etkilerini dikkatlice gözlemlemektir. Hata senaryolarına verilen tepkilerin olası senaryolar olabileceğini bilmek ve bu çıktılara göre hareket etmek gerekmektedir. Pod tabanlı kaos deneyleri hizmet sürekliliğini bozarken, stres veya kapasiteye dayalı senaryolar hizmetler arasındaki iletişimdeki eksiklikleri ortaya çıkarmıştır. Bazı hizmetlerin yüksek yük koşullarında işlevlerini yerine getiremediği gözlemlenmiştir. Bu şekilde, mevcut ölçeklendirme ayarlarının yeterli olmadığı belirlenmiştir. Bu çıktılar sayesinde, üretim ortamındaki mevcut yapılandırmamızı yeniden düzenlemeye karar verdik. Sistemin dayanıklılığını artırmak için yapılan ayarlardan sonra, kaos senaryolarını tekrarlamak, yapılan iyileştirmelerin sistem üzerindeki etkisini ölçer. Aynı senaryolara sistemin verdiği tepkiler, revize edilmiş versiyonla birlikte gözlemlenmelidir. Doğru yapılandırmalar, mikroservis mimarisinin geliştirilmiş dayanıklılığını gösterecektir.

Bu çalışma içerisinde, Güvenlik ve Dayanıklılık için Senaryo Tabanlı Kaos Mühendisliğinin Simülasyonunu ele aldık. Bu tez için Kaos Mühendisliği veya Hata Enjeksiyonu konularını ele alan birçok makale incelendi. Mikroservis

mimarisine uygun ve sistem stabilizasyonuna dayalı araştırmalara yer verildi. Sistemimiz için en uygun makaleler toplandı ve en yaygın kullanılan araçlar belirlendi. Chaos Mesh ve Litmus araçları seçildi, çeşitli hata koşulları simüle edildi ve sistemimizin stres altındaki dayanıklılığı gözlemlendi. Gözlemler sonucunda, sistemin eksiklikleri keşfedildi. Bu senaryoların otomatikleştirilmesinin ve sistemin sürekli izlenmesinin önemi anlaşıldı.

Gelecek çalışmalarda, üretim ortamlarına uyarlanabilecek kaos deneyleri için daha yenilikçi metodolojiler geliştirilebilir. Son kullanıcı deneyimi üzerindeki etkiyi en aza indirmek ve bu kaos deneylerini üretim ortamlarında otomatikleştirmek gerekebilir. Bu sayede, sistem dayanıklılığı sürekli olarak iyileştirilebilir. Ayrıca, ileri araştırmalar, kaos koşulları altında sistem davranışını tahmin etmek ve potansiyel hataları azaltmak için sistem yapılandırmalarını dinamik olarak ayarlamak amacıyla gelişmiş makine öğrenimi algoritmalarının entegrasyonunu da araştırabilir.

KAYNAKLAR

Al-Kuwaiti, M., Kyriakopoulos, N., & Hussein, S. (2009). A comparative analysis of network dependability, fault-tolerance, reliability, security, and survivability. *IEEE Communications Surveys & Tutorials*, 11(2), 106-124.

Addeen, H. H. (2019). *A Dynamic Fault Tolerance Model for Microservices Architecture*. South Dakota State University.

Angerstein, T. (2018). *Modularization of representative load tests for microservice applications* (Master's thesis).

Basiri, A., Hochstein, L., Jones, N., & Tucker, H. (2019, May). Automating chaos experiments in production. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)* (pp. 31-40). IEEE.

Camacho, C., Cañizares, P. C., Llana, L., & Núñez, A. (2022). Chaos as a Software Product Line—a platform for improving open hybrid-cloud systems resiliency. *Software: Practice and Experience*, 52(7), 1581-1614.

Frank, S., Hakamian, M. A., Wagner, L., Kesim, D., von Kistowski, J., & van Hoorn, A. (2021). Scenario-based Resilience Evaluation and Improvement of Microservice Architectures: An Experience Report. In *ECSA (Companion)*.

Herscheid, L., Richter, D., & Polze, A. (2015). Experimental assessment of cloud software dependability using fault injection. *Springer*, 121-128.

Jernberg, H. (2020). *Building a Framework for Chaos Engineering*. LU-CS-EX.

Jernberg, H., Runeson, P., & Engström, E. (2020, October). Getting Started with Chaos Engineering-design of an implementation framework in practice. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 1-10).

Kesim, D. (2019). *Assessing resilience of software systems by the application of chaos engineering: a case study* (Bachelor's thesis).

Lenka, R. K., Padhi, S., & Nayak, K. M. (2018, October). Fault Injection Techniques-A Brief Review. In *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)* (pp. 832-837). IEEE.

Lion Wagner, S. F., Hakamian, A., & van Hoorn, A. (2021). MiSim—A Lightweight and Extensible Simulator for a Scenario-Based Resilience Evaluation of Microservice Architectures.

Petersson, L. (2022). *An Empirical Investigation of the Acceptance of Chaos Engineering*.

Torkura, K. A., Sukmana, M. I., Cheng, F., & Meinel, C. (2020). Cloudstrike: Chaos engineering for security and resiliency in cloud infrastructure. *IEEE Access*, 8, 123044-123060.

Zorn, C. (2021). *Interactive elicitation of resilience scenarios in microservice architectures* (Master's thesis).