



**T.C.
MARMARA ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

Kaos Mühendisliği: Prensipler ve Pratikler Proje Raporu

**Hazırlayan
Halil İbrahim KAVALCI 170424823**

İçindekiler

1. Giriş.....	3
2. Proje Tanımı ve Oluşturulan Mikroservisler.....	3
2.1 Frontend Servisi.....	3
2.2 Authentication Servisi.....	5
2.3 Product Servisi.....	5
2.4 Checkout Servisi.....	6
2.5 Payment Servisi.....	7
3. Kaos Deneyleri.....	8
3.1 Network Delay Deneyi.....	8
3.2 HTTP Response Replacement Deneyi.....	9
3.3 Pod Failure Deneyi.....	10
3.4 CPU Stress Deneyi.....	11
3.5 DNS Chaos Deneyi.....	12
4. Sonuç.....	13

1. Giriş

Bu proje, mikroservis mimarisi ile geliştirilen bir e-ticaret uygulaması simülasyonu üzerinde kaos mühendisliği deneyleri gerçekleştirerek sistemin farklı hata senaryolarına verdiği tepkilerin gözlemlenmesi amacıyla hazırlanmıştır.

Proje kapsamında uygulama, Kubernetes ortamında çalışan mikroservisler şeklinde tasarlanmıştır. Kaos deneyleri Chaos Mesh aracı kullanılarak uygulanmış, deney sonuçları ise Postman aracılığıyla gözlemlenmiştir. Bu deneylerin temel amacı; sistemin hata durumlarındaki davranışını analiz etmek, olası zayıf noktaları tespit etmek ve sistemin dayanıklılığını artırmaya yönelik iyileştirme önerileri sunmaktır.

2. Proje Tanımı ve Oluşturulan Mikroservisler

Geliştirilen uygulama, temel bir e-ticaret senaryosunu simüle eden mikroservis tabanlı bir mimariye sahiptir. Sistem; kullanıcıların kimlik doğrulama işlemlerini gerçekleştirebildiği, ürünleri görüntüleyebildiği ve sipariş oluşturabildiği bir yapı olarak tasarlanmıştır. Uygulamanın geliştirilmesinde Python tabanlı Flask framework'ü kullanılmıştır.,

2.1 Frontend Servisi

Kullanıcıdan gelen HTTP isteklerini alır, ilgili backend servislerine yönlendirir ve dönen cevabı kullanıcıya iletir. Beklenmeyen durumlarda kullanıcıyı bilgilendirir.

Endpoints

Index

```
@app.route('/')
def index():
    try:
        # Check Auth Service Health
        auth_health = requests.get(f'{AUTH_SERVICE_URL}/health').json()

        # Check Product Service Health
        product_health = requests.get(f'{PRODUCT_SERVICE_URL}/health').json()

        # Get Products
        start_time = time.time()
        products_response = requests.get(f'{PRODUCT_SERVICE_URL}/products')
        response_time = time.time() - start_time

        products = products_response.json().get('products') if products_response.status_code == 200 else []

        notifications = []
        if response_time > 2.0: # Threshold for "high load"
            notifications.append("System is experiencing high load, please be patient.")

        for product in products:
            if product.get('replaced'):
                notifications.append(f"Warning: Price for {product['name']} has been replaced!")

        return jsonify({
            'frontend status': 'running',
            'auth service health': auth_health,
            'product service health': product_health,
            'products available': products,
            'notifications': notifications
        })
    except requests.exceptions.ConnectionError as e:
        return jsonify({'error': f'Service connection failed: {str(e)}'}, 503)
```

Bu endpoint, hem Authentication hem de Product servislerinin sağlık durumunu kontrol eder, ürün listesini alır, yüksek yük veya fiyat değişiklikleri gibi durumlar için bildirimler oluşturur ve tüm bilgileri JSON olarak döner.

Checkout

```
@app.route('/checkout', methods=['POST'])
def checkout():
    try:
        checkout_response = requests.post(f'{CHECKOUT_SERVICE_URL}/checkout', json={})
        checkout_data = checkout_response.json()

        if checkout_response.status_code == 503 and 'Auth service' in checkout_data.get('error', ''):
            return jsonify({'error': 'Checkout failed: Auth service is down. Please try again later.'}), 503

        return jsonify(checkout_data), checkout_response.status_code
    except requests.exceptions.ConnectionError as e:
        return jsonify({'error': f'Service connection failed: {str(e)}'}), 503
```

Bu endpoint, Checkout servisine bir POST isteği göndererek ödeme işlemini başlatır ve Checkout veya Auth servisiyle ilgili hataları kontrol ederek sonucu JSON formatında döner.

Get Products

```
@app.route('/products', methods=['GET'])
def get_products():
    try:
        products_response = requests.get(f'{PRODUCT_SERVICE_URL}/products')
        return jsonify(products_response.json()), products_response.status_code
    except requests.exceptions.ConnectionError as e:
        return jsonify({'error': f'Service connection failed: {str(e)}'}), 503
```

Bu endpoint, gelen GET isteğiyle Product servisten ürün verilerini alır ve geri döner. Bağlantı hatası durumunda 503 hata koduyla bir hata mesajı döner.

Get Product By ID

```
@app.route('/products/<product_id>', methods=['GET'])
def get_product_by_id(product_id):
    try:
        products_response = requests.get(f'{PRODUCT_SERVICE_URL}/products/{product_id}')
        if(products_response.json().get('replaced')):
            return jsonify({'error': 'Product price has been replaced', 'product': products_response.json()}), 500
        return jsonify(products_response.json()), products_response.status_code
    except requests.exceptions.ConnectionError as e:
        return jsonify({'error': f'Service connection failed: {str(e)}'}), 503
```

Bu endpoint, verilen ürün ID'siyle Product servisten ürün bilgilerini çekip hata durumlarını uygun HTTP kodlarıyla dönen bir GET isteğini işler.

Login

```
@app.route('/login', methods=['POST'])
def login():
    try:
        data=request.json
        login_response = requests.post(f'{AUTH_SERVICE_URL}/login', json={'username': data['username'], 'password': data['password']})
        login_data = login_response.json()
        if(login_data['timestamp'] > time.time() + 1):
            return jsonify({'error': 'Time drift detected'}), 401
        return jsonify({'token': login_data['token'], 'message': 'Login successful','timestamp_auth': login_data['timestamp'], 'timestamp_front': time.time()}), login_response.status_code
    except Exception as e:
        return jsonify({'error': f'Service connection failed: {str(e)}'}), 503
```

Bu endpoint, kullanıcıdan gelen POST isteğiyle Auth servisine giriş bilgilerini iletip token dönerek login işlemini gerçekleştirir.

2.2 Authentication Servisi

Kullanıcı kimlik doğrulama işlemlerini gerçekleştirir. Giriş ve yetkilendirme kontrolleri bu servis üzerinden yapılır.

Endpoints

Health Check

```
@app.route('/health', methods=['GET'])
def health_check():
    return jsonify({'status': 'healthy', 'service': 'auth-service'}), 200
```

Bu endpoint, servisin sağlık durumunu döner.

Login

```
@app.route('/login', methods=['POST'])
def login():
    # Mock login
    data = request.json
    username = data.get('username')
    password = data.get('password')

    if username == 'admin' and password == 'password':
        return jsonify({'token': 'mock-token-12345', 'message': 'Login successful', 'timestamp': time.time()}), 200
    return jsonify({'message': 'Invalid credentials', 'timestamp': time.time()}), 401
```

Bu endpoint, mock login işlemini gerçekleştirerek geriye mock token döner.

2.3 Product Servisi

Ürün bilgilerini yönetir. Ürün listeleme, ürün detaylarını getirme gibi işlemlerden sorumludur.

Endpoints

Health Check

```
@app.route('/health', methods=['GET'])
def health_check():
    return jsonify({'status': 'healthy', 'service': 'product-service'}), 200
```

Bu endpoint, servisin sağlık durumunu döner.

Get Products

```
@app.route('/products', methods=['GET'])
def get_products():
    return jsonify({'products': products}), 200
```

Bu endpoint, ürün bilgilerini döner.

Get Product By ID

```
@app.route('/products/<int:product_id>', methods=['GET'])
def get_product_by_id(product_id):
    for product in products:
        if product['id'] == product_id:
            return jsonify(product), 200
    return jsonify({'error': 'Product not found'}), 404
```

Bu endpoint, verilen ID numarasına göre bir ürün döner.

2.4 Checkout Servis

Kullanıcıların sipariş oluşturma süreçlerini yönetir.

Endpoints

Health Check

```
@app.route('/health', methods=['GET'])
def health_check():
    return jsonify({'status': 'healthy', 'service': 'checkout-service'}), 200
```

Bu endpoint, servisin sağlık durumunu döner.

Checkout

```
@app.route('/checkout', methods=['POST'])
def checkout():
    data = request.json
    # Mock checkout logic
    amount = 100.0 # Mock amount

    payment_service_url = os.environ.get('PAYMENT_SERVICE_URL', 'http://payment-service:5003')
    auth_service_url = os.environ.get('AUTH_SERVICE_URL', 'http://auth-service:5001')

    # Check Auth Service
    try:
        auth_response = requests.get(f'{auth_service_url}/health')
        if auth_response.status_code != 200:
            return jsonify({'error': 'Auth service unhealthy'}), 503
    except requests.exceptions.RequestException:
        return jsonify({'error': 'Auth service unreachable'}), 503

    start_time = time.time()
    try:
        payment_response = requests.post(f'{payment_service_url}/pay', json={'amount': amount})
        payment_data = payment_response.json()
    except requests.exceptions.RequestException as e:
        return jsonify({'error': 'Payment service unreachable', 'details': str(e)}), 503
    end_time = time.time()

    response_time = end_time - start_time

    if response_time > 4:
        return jsonify({'error': 'Payment processing timed out', 'details': 'Response time: ' + str(response_time)}), 504

    return jsonify({
        'status': 'success',
        'message': 'Checkout processed successfully',
        'order_id': 'order-12345',
        'payment_response': payment_data,
        'payment_response_time': response_time
    }), 200
```

Bu endpoint, gelen POST isteğinde Payment ve Authentication servislerini kontrol ederek ödeme işlemini başlatır, servislerin erişilebilirliğini ve yanıt süresini denetler, 4 saniyeyi aşan gecikmelerde timeout hatası döner ve başarılı durumda sipariş bilgileriyle birlikte ödeme sonucunu JSON olarak geri verir.

2.5 Payment Servisi

Ödeme işlemlerini gerçekleştirir.

Endpoints

Health Check

```
@app.route('/health', methods=['GET'])
def health_check():
    return jsonify({'status': 'healthy', 'service': 'payment-service'}), 200
```

Bu endpoint, servisin sağlık durumunu döner.

Process Payment

```
@app.route('/pay', methods=['POST'])
def process_payment():
    data = request.json
    amount = data.get('amount')

    return jsonify({
        'status': 'success',
        'message': 'Payment processed successfully',
        'amount': amount
    }), 200
```

Bu endpoint, gelen POST isteğinde gönderilen tutarı alıp başarılı ödeme mesajıyla birlikte JSON formatında geri dönen basit bir ödeme işleme servisini temsil eder.

3. Kaos Deneyleri

3.1 Network Delay Deneyi

Amaç:

Payment Service'den gelen response'un gecikmeli gelmesinin Checkout Service üzerindeki etkisini gözlemlemek.

Uygulama:

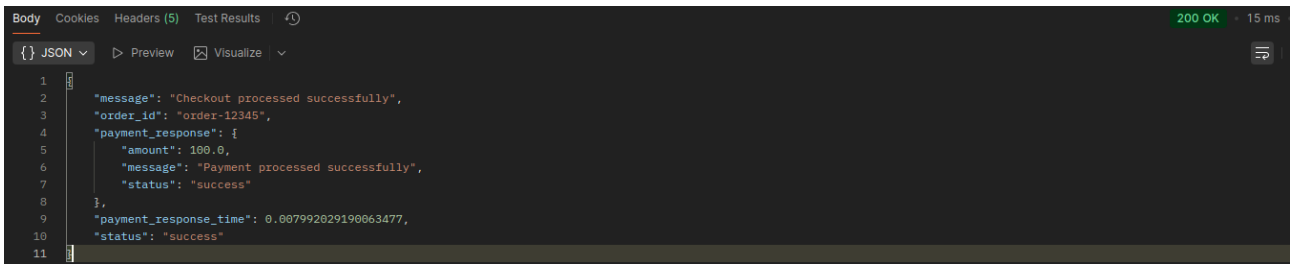
Payment Service'den gelen response'a bir gecikme eklenmiştir. Frontend'in /checkout endpoint'ine POST isteği gönderilmiştir.

```
apiVersion: chaos-mesh.org/v1alpha1
kind: NetworkChaos
metadata:
  name: payment-network-delay
  namespace: chaos-project
spec:
  action: delay
  mode: all
  selector:
    namespaces:
      - chaos-project
    labelSelectors:
      app: payment-service
  delay:
    latency: "5000ms"
    correlation: "100"
    jitter: "100ms"
    duration: "5m"
```

Network Delay Injection Yaml Dosyası

Gözlem:

Payment Service'den dönen response belirli bir süreyi aştığı için Checkout Service, Frontend'e bilgi iletmıştır.



Network Delay Injection Öncesi



Network Delay Injection Sonrası

3.2 HTTP Response Replacement Deneyi

Amaç:

Product Service'den dönen response değiştirilerek Frontend'in davranışını gözlemlemek.

Uygulama:

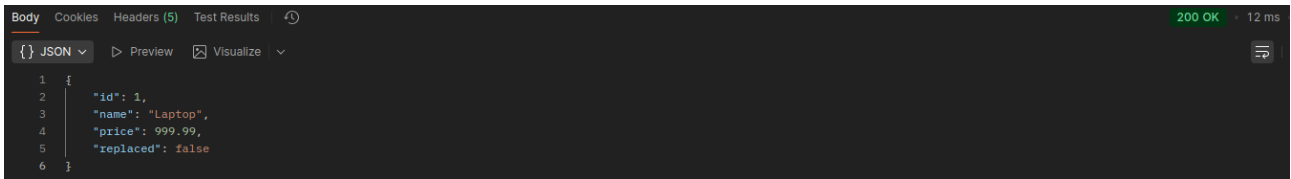
Product Service'den gelen response değiştirilmiştir.

```
apiVersion: chaos-mesh.org/v1alpha1
kind: HTTPChaos
metadata:
  name: product-price-patch
  namespace: chaos-project
spec:
  mode: all
  selector:
    labelSelectors:
      app: product-service
  target: Response
  port: 5002
  method: GET
  path: /products/1
  patch:
    body:
      type: JSON
      value: '{"id": 1, "name": "Laptop", "price": 0, "replaced": true}'
  duration: "5m"
```

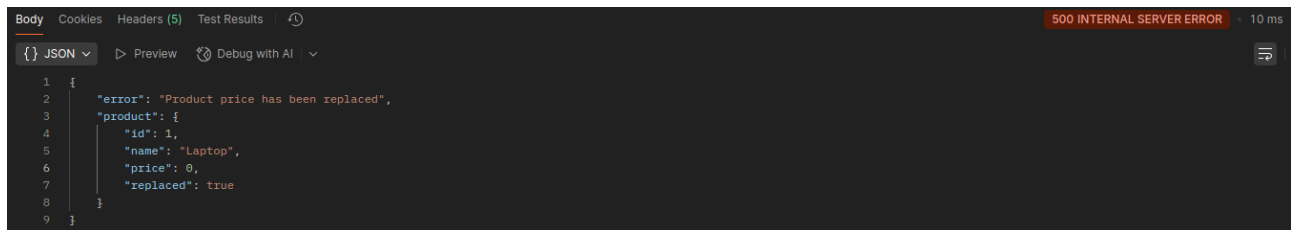
HTTP Response Replacement Injection Yaml Dosyası

Gözlem:

Product Service'den dönen response'un değiştiği Frontend tarafından yakalanıp kullanıcıya bilgi verilmiştir.



HTTP Response Replacement Injection Öncesi



HTTP Response Replacement Injection Sonrası

3.3 Pod Failure Deneyi

Amaç:

Checkout sırasında Auth Service çökmesi durumunda sistem nasıl davrandığını gözlemlemek.

Uygulama:

Auth Service'in pod'u 30 saniye boyunca fail durumuna düşmüştür.

```
apiVersion: chaos-mesh.org/v1alpha1
kind: PodChaos
metadata:
  name: pod-kill-auth
  namespace: chaos-project
spec:
  action: pod-failure
  mode: one
  selector:
    namespaces:
      - chaos-project
    labelSelectors:
      app: auth-service
  duration: "30s"
```

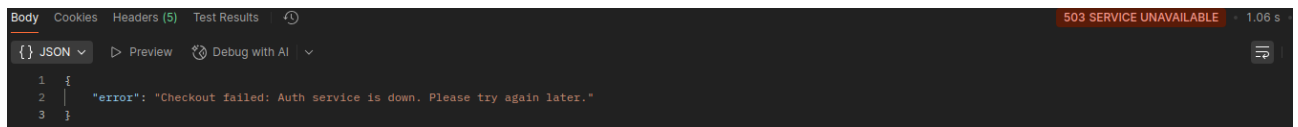
Şekil 7 – Pod Failure Injection Yaml Dosyası

Gözlem:

Auth Service'den response alınamadığı için kullanıcıya bilgi verilmiştir.



Şekil 8 – Pod Failure Injection Öncesi



Şekil 9 – Pod Failure Injection Sonrası

3.4 CPU Stress Deneyi

Amaç:

CPU kullanımı %100 olduğu zaman sistemin davranışını gözlemlemek.

Uygulama:

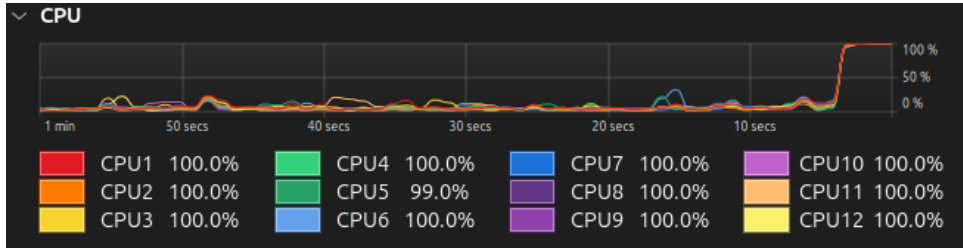
CPU kullanımı %100'e çıkarılmıştır.

```
apiVersion: chaos-mesh.org/v1alpha1
kind: StressChaos
metadata:
  name: total-meltdown
  namespace: chaos-project
spec:
  mode: all
  selector:
    namespaces:
      - chaos-project
    labelSelectors:
      app: product-service
  stressors:
    cpu:
      workers: 12
      load: 100
      duration: "5m"
```

Şekil 10 – CPU Stress Injection Yaml Dosyası

Gözlem:

Sistemde çok düşük bir gecikme artışı görülmüştür.



Şekil 11 – CPU Stress Injection Sonrası System Monitor

```
Body Cookies Headers (5) Test Results 200 OK 27 ms
JSON Preview Visualize
1
2 "auth_service_health": {
3   "service": "auth-service",
4   "status": "healthy"
5 },
6 "frontend_status": "running",
7 "notifications": [],
8 "product_service_health": {
9   "service": "product-service",
10  "status": "healthy"
11 },
12 "products_available": [
13   {
14     "id": 1,
15     "name": "Laptop",
16     "price": 999.99,
17     "replaced": false
18   },
19   {
20     "id": 2,
21     "name": "Smartphone",
22     "price": 499.99,
23     "replaced": false
24   },
25   {
26     "id": 3,
27     "name": "Headphones",
28     "price": 79.99,
29     "replaced": false
30   }
31 ]
32
```

Şekil 12 – CPU Stress Injection Sonrası 27 ms Gecikme

3.5 DNS Chaos Deneyi

Amaç:

Frontend'in yanlış IP adreslere request göndermesi sonucu sistemin davranışının gözlemlenmesi.

Uygulama:

Chaos Mesh DNSChaos kullanılarak Frontend'in, Auth Service yerine rastgele bir IP adresine istek atması sağlanmıştır.

```
apiVersion: chaos-mesh.org/v1alpha1
kind: DNSChaos
metadata:
  name: dns-random-fail
  namespace: chaos-project
spec:
  action: random
  mode: all
  selector:
    namespaces:
      - chaos-project
    labelSelectors:
      app: frontend
  patterns:
    - "auth-service"
  duration: "5m"
```

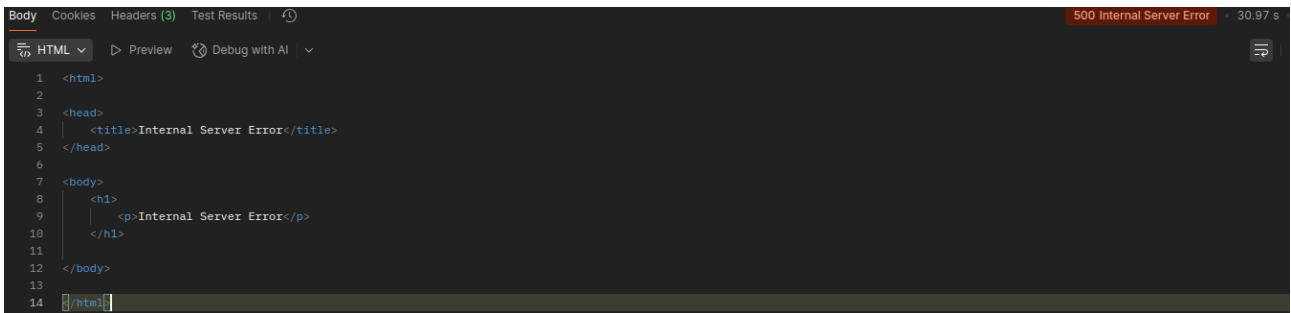
Şekil 13 – DNS Chaos Injection Yaml Dosyası

Gözlem:

Yanlış IP adresten yanıt gelmediği için Server Error dönmüştür.



Şekil 14 – DNS Chaos Injection Öncesi



Şekil 15 – DNS Chaos Injection Sonrası

4. Sonu

Bu proje kapsamında gerekleřtirilen kaos deneyleri, mikroservis mimarisine sahip bir e-ticaret uygulamasının gerek dnya hata senaryolarına karřı ne kadar dayanıklı olduėunu ortaya koymuřtur. Elde edilen sonular, sistemin bazı durumlarda otomatik olarak toparlanabildiėini, ancak bazı senaryolarda kullanıcı deneyiminin olumsuz etkilendiėini gstermiřtir.

Kaos Mhendisliėi yaklařımı sayesinde sistemdeki zayıf noktalar erken ařamada tespit edilmiř ve iyileřtirme yapılabilir alanlar belirlenmiřtir. Bu alıřma, daėıtık sistemlerde gvenilirlik ve dayanıklılıėın artırılması aısından nemli bir pratik deneyim sunmuřtur.