Teknik Tasarım Raporu: Dynatrace API v2 ile Otomatik Servis Topolojisi Çıkarma

Teknik Tasarım Raporu: Dynatrace API v2 ile Otomatik Servis Topolojisi Çıkarma

1. Yönetici Özeti (Executive Summary)

Bu rapor, finansal bir kurumun dinamik ve karmaşık mikroservis ortamında servisler arası bağımlılıkların otomatik olarak çıkarılması ve haritalanması için tasarlanmış teknik stratejiyi ve mimariyi özetlemektedir. Site Reliability Engineering (SRE) ekipleri için, sürekli değişen bu topolojiyi manuel olarak takip etmek sürdürülemez bir operasyonel yüktür. Bu nedenle, otomasyon, yalnızca bir verimlilik artışı değil, aynı zamanda hata analizi, etki değerlendirmesi ve uyumluluk denetimi gibi temel SRE fonksiyonları için kritik bir yetenek haline gelmiştir.

Projenin temel amacı, servisler arası bağımlılıkları (Service Flow) manuel kontrollere dayalı anlık görüntüler yerine, Dynatrace API'sini kullanarak programatik, sürekli ve tekrarlanabilir bir süreçle çıkarmak ve görselleştirmektir. Bu süreç, güncel ve doğru bir topoloji haritasının her an SRE ekiplerinin ve diğer paydaşların kullanımına sunulmasını sağlayacaktır.

Bu görevi gerçekleştirmek için, resmi olarak kullanımdan kaldırılmış (deprecated) olan eski Smartscape API'si yerine modern **Monitored Entities API v2** (`GET /api/v2/entities`) endpoint'inin kullanılması, risk yönetimi açısından zorunlu bir mimari karardır. Desteklenmeyen bir API kullanmak, önemli bir bakım riski ve teknik borç birikimi yarattığından, API v2, bu görev için tek desteklenen ve doğru yoldur. Sunduğu güçlü `entitySelector` filtreleme yetenekleri, `fields` parametresi ile veri yükünü (payload) optimize etme imkanı ve uzun vadeli platform desteği ile kurumsal düzeyde bir otomasyon için gerekli olan ölçeklenebilirliği ve sürdürülebilirliği sağlar.

Bu otomasyonun hayata geçirilmesi, SRE ekibine önemli ölçüde zaman ve efor kazandıracaktır. Topoloji verilerinin otomatik olarak çıkarılması; hata kök neden analizi (RCA), planlanan bir değişikliğin potansiyel etkisini değerlendirme (impact analysis) ve Yapılandırma Yönetimi Veritabanı (CMDB) senkronizasyonu gibi kritik süreçleri doğrudan iyileştirecektir.

Bu özetin ardından raporun devamında, bu stratejik hedefe ulaşmak için gereken teknik mimari, uygulama mantığı ve kurumsal düzeyde operasyonel gereksinimler detaylı bir şekilde ele alınacaktır.

---

2. Teknik Mimari ve API Stratejisi

Başarılı bir otomasyonun temeli, doğru API endpoint'inin ve parametrelerinin seçilmesine dayanır; bu, projenin en temel mimari kararıdır. Yanlış bir API stratejisi, yalnızca verimsizliğe yol açmakla kalmaz, aynı zamanda gelecekte yönetilmesi zor teknik borçlar yaratır. Bu nedenle, bu proje için seçilen API stratejisi, ölçeklenebilirlik, verimlilik ve uzun vadeli sürdürülebilirlik ilkeleri üzerine kurulmuştur.

Hedef API Endpoint'i

Otomasyon betiğinin kullanacağı tek ve merkezi endpoint, Monitored Entities API v2'dir.

- **HTTP Metodu ve URL:** `GET /api/v2/entities`

Bu endpoint, `HOST`, `SERVICE`, `PROCESS_GROUP_INSTANCE` gibi farklı varlık türlerini sorgulamak için birleşik bir erişim noktası sunar. Bu standart yapı, API istemcilerinin (client) geliştirme ve bakım süreçlerini önemli ölçüde basitleştirir, çünkü her varlık türü için ayrı bir endpoint öğrenmek ve kodlamak gerekmez.

Filtreleme Mantığı

API'den yalnızca hedefe yönelik ve yönetilebilir bir veri seti çekmek için `entitySelector` parametresi kritik bir rol oynar. Bu parametre, sorgu kapsamını daraltarak ağ trafiğini ve işleme yükünü en aza indirir.

Parametre	Değer	Amaç ve Stratejik Gerekçe

`entitySelector`	`type("SERVICE")`	Yanıtın yalnızca servis tipi varlıkları içermesini sağlayarak veri setini hedefe odaklı ve yönetilebilir hale getirir.
`entitySelector`	`mzId("...")`	Finansal hizmetler gibi sıkı regülasyonlara tabi ortamlarda, veri erişimini ve topoloji kapsamını PCI, SOX gibi uyumluluk sınırlarına göre kesin olarak ayırmak için zorunludur. Bu, denetim (audit) takibini kolaylaştırır ve yetkisiz servis etkileşim riskini en aza indirir.

Veri Alanı Seçimi

Topoloji haritasının temelini oluşturan servisler arası bağımlılıkları elde etmek için, API yanıtında hangi veri alanlarının döndürüleceğini `fields` parametresi ile açıkça belirtmek gerekir. Bu, gereksiz veri transferini önleyerek performansı optimize eder.

- **Gerekli Parametre:** `fields=+fromRelationships.calls,+toRelationships.called_by`

Bu iki alan, bir servisin topolojideki yerini ve bağlantılarını tanımlayan yönlü ilişkileri temsil eder. Anlamsal olarak farkları şöyledir:

- `fromRelationships.calls`: Sorgulanan servisin yaptığı **giden** çağrıları temsil eder. Bu, "kaynak" (sorgulanan servis) ile "hedef" (çağrılan servis) arasındaki ilişkiyi gösterir (Kaynak → Hedef).
- `toRelationships.called_by`: Sorgulanan servise yapılan **gelen** çağrıları temsil eder. Bu, "kaynak" (çağıran servis) ile "hedef" (sorgulanan servis) arasındaki ilişkiyi gösterir (Hedef ← Kaynak).

Bu mimari temeller, binlerce servisin bulunduğu büyük ölçekli bir ortamda veriyi verimli, hatasız ve eksiksiz bir şekilde çekmek için gereken temel uygulama mantığına zemin hazırlamaktadır.

---

3. Kritik Uygulama Mantığı (Core Implementation Logic)

Teorik API bilgisini, binlerce servisin bulunduğu büyük bir bankacılık ortamında ölçeklenebilir ve hatasız çalışan bir otomasyon betiğine dönüştürmek, doğru

algoritmaların ve uygulama mantığının benimsenmesini gerektirir. Bu bölüm, bu dönüşüm için gereken temel teknikleri detaylandırmaktadır.

Sayfalama (Pagination) Algoritması

Yüksek hacimli ortamlarda, tüm servis verilerini tek bir API çağrısı ile çekmek mümkün değildir. Verinin eksiksiz ve verimli bir şekilde alınması için imleç tabanlı (cursor-based) sayfalama mekanizması kullanılmalıdır.

- **Sayfa Boyutu Optimizasyonu:** API çağrı sayısını en aza indirmek için `pageSize` parametresinin yüksek bir değere ayarlanması tavsiye edilir. Monitored Entities API dokümantasyonu varsayılan değeri 50 olarak belirtse de, `pageSize=500` değeri, Metrics API gibi diğer v2 endpoint'leri tarafından da teyit edilen ve çağrı hacmini minimize etmek için yerleşik bir en iyi uygulamadır.
- **İmleç Mekanizması:** Sayfalama mantığı şu şekilde işler:
  1. İlk istek, `entitySelector` ve `fields` gibi tüm filtreleme parametreleriyle birlikte yapılır.
  2. API yanıtı, sonuçların bir sonraki sayfasını işaret eden bir `nextPageKey` imleci içeriyorsa bu değer saklanır.
  3. Bir sonraki sayfanın verisini çekmek için, yalnızca bu `nextPageKey` değeri kullanılarak yeni bir API isteği yapılır. Bu döngü, API yanıtında artık bir `nextPageKey` dönmeyene kadar devam eder.

KRİTİK UYARI: `nextPageKey` ile yapılan sonraki sayfa isteklerinde, ilk istekte kullanılan `entitySelector`, `fields` ve `pageSize` gibi diğer tüm sorgu parametrelerinin KESİNLİKLE gönderilmemesi gerekir. Bu kurala uyulmaması, API'nin isteği geçersiz kabul etmesine ve `HTTP 400 Bad Request` hatası döndürmesine neden olur. Bu, Dynatrace API v2'nin sayfalama mekanizmasının temel bir kuralıdır.

Veri Ayrıştırma (Parsing) Süreci

API yanıtından gelen JSON verisindeki `relationships` nesnesi, bağımlılıkların çıkarılması için dikkatle işlenmesi gereken iç içe geçmiş (nested) bir yapıya sahiptir.

- **İç İçe Yapıyı Anlama:** Bir servisin başka bir servise olan bağımlılığını bulmak için, betiğin yalnızca `relationships.calls` nesnesini değil, onun içindeki `SERVICE`

anahtarına sahip diziyi işlemesi gerekir. Örneğin, gerçek hedef servis kimlikleri `relationships.calls.SERVICE` ve `relationships.called_by.SERVICE` dizilerinde yer alır.

- **Kenar Listesine (Edge List) Dönüştürme:** Otomasyon betiğinin temel görevi, bu iç içe geçmiş API yanıtını, evrensel olarak kullanılabilir bir "Kaynak Servis ID → Hedef Servis ID" Kenar Listesi'ne dönüştürmektir. Bu işlem şu adımları izlemelidir:
  1. Sayfalama döngüsünden gelen her bir servis nesnesi için, `entityId` değerini alın. Bu, mevcut döngüdeki ana servisimizdir.
  2. **Giden Bağımlılıkları İşleyin:** Servis nesnesindeki `fromRelationships.calls.SERVICE` dizisinin varlığını kontrol edin. Eğer varsa, bu dizideki her bir hedef ID için:
     - `(Mevcut Servis ID)`'yi **Kaynak** olarak ve `(Hedef ID)`'yi **Hedef** olarak alarak bir kenar (edge) oluşturun.
  3. **Gelen Bağımlılıkları İşleyin:** Servis nesnesindeki `toRelationships.called_by.SERVICE` dizisinin varlığını kontrol edin. Eğer varsa, bu dizideki her bir kaynak ID için:
     - `(Kaynak ID)`'yi **Kaynak** olarak ve `(Mevcut Servis ID)`'yi **Hedef** olarak alarak bir kenar (edge) oluşturun. Bu işlem, her servis için tekrarlandığında, tüm topolojiyi temsil eden eksiksiz bir Kenar Listesi ortaya çıkarır. Bu düzleştirilmiş format, CMDB sistemleri, grafik veritabanları veya görselleştirme araçları için evrensel bir girdi görevi görür.

Bu temel uygulama mantığı, kurumsal düzeyde güvenilir ve güvenli bir şekilde çalışabilmesi için, bir sonraki bölümde ele alınacak olan operasyonel standartlarla desteklenmelidir.

---

4. Operasyonel Mükemmellik ve Güvenlik (Enterprise Readiness)

Geliştirilen otomasyon betiğini bir "proof-of-concept" seviyesinden, bir bankanın üretim ortamının gerektirdiği yüksek güvenlik, dayanıklılık ve ölçeklenebilirlik standartlarına taşımak, operasyonel mükemmellik ilkelerinin uygulanmasını gerektirir. Bu bölüm, betiğin kurumsal düzeyde hazır hale getirilmesi için kritik olan hata yönetimi, API limitleri ve güvenlik stratejilerini ele almaktadır.

Hata Yönetimi ve API Limitleri

Dynatrace API'si, platformun kararlılığını korumak için istek limitleri (rate limits) uygular. Bu limitlerin akıllıca yönetilmesi, otomasyonun kesintisiz çalışması için zorunludur.

- **API Erişim Limiti:** Dynatrace SaaS ortamları için genel API erişim limiti dakikada **50 istektir (RPM)**.
- **Hata Kodu:** Bu limite ulaşıldığında, API `HTTP 429 Too Many Requests` yanıt koduyla istekleri geçici olarak reddeder.
- **En İyi Strateji: İki Seviyeli Senkronize Geri Çekilme (Synchronized Backoff):** Üretim ortamı için geliştirilen betikler, sabit bir bekleme süresi veya genel bir "Exponential Backoff" stratejisi yerine, API tarafından sağlanan HTTP yanıt başlıklarını okuyan daha akıllı, iki seviyeli bir mekanizma uygulamalıdır:
    1. **Proaktif Duraklatma:** Betik, her API yanıtında `X-RateLimit-Remaining` başlığını izlemelidir. Kalan istek sayısı, yapılandırılabilir bir güvenlik eşiğinin (örneğin, 5) altına düşerse, betik, `X-RateLimit-Reset` başlığında belirtilen Unix zaman damgasına kadar proaktif olarak yürütmeyi duraklatmalıdır.
    2. **Reaktif Geri Çekilme:** Eğer (örneğin eş zamanlı çalışan başka istemciler nedeniyle) beklenmedik bir `HTTP 429` hatası alınırsa, betik reaktif olarak devreye girmeli ve yine `X-RateLimit-Reset` zaman damgasına kadar beklemelidir. Bu senkronize ve proaktif yaklaşım, API kotasının en verimli şekilde kullanılmasını sağlar ve otomasyonun dayanıklılığını artırır.

Güvenlik ve Yetkilendirme Stratejisi

Finansal bir kurumda API entegrasyonlarının güvenliği taviz verilemez bir önceliktir. Bu nedenle, API token yönetimi ve yetkilendirme sıkı güvenlik prensiplerine dayanmalıdır.

- **API Token Kapsamı (Scope):** Bu otomasyon için gereken minimum yetki kapsamı, **En Az Ayrıcalık İlkesi (Principle of Least Privilege - PoLP)** ile uyumlu olarak `entities.read`'dir. Bu kapsam, varlık verilerini okumak için yeterlidir. Güvenlik riskini en aza indirmek için, `entities.write` gibi yazma yetkileri veya gereksiz diğer okuma yetkileri **kesinlikle verilmemelidir**.

- **Token Yönetimi:** API token'ları, parola gibi hassas bilgilerdir ve **kesinlikle kaynak koduna veya düz metin yapılandırma dosyalarına yazılmamalıdır**. Bu, temel bir güvenlik ihlalidir. Kurumsal standartlara uygun olarak, token'lar çalışma zamanında güvenli bir şekilde alınmalıdır. Bunun için en iyi uygulamalar şunlardır:
  - **Ortam Değişkenleri (Environment Variables):** Basit ve etkili bir yöntemdir.
  - **Kurumsal Sır Yönetim Aracı (Secrets Management Tool):** HashiCorp Vault, Azure Key Vault veya AWS Secrets Manager gibi merkezi sistemler, token'ların güvenli bir şekilde saklanması, erişiminin denetlenmesi ve düzenli olarak döndürülmesi (rotate) için en güvenli ve ölçeklenebilir çözümdür.

Raporun bu teknik detayları ve operasyonel gereksinimleri, projenin başarısı için en kritik önerileri sunacak olan sonuç bölümüne zemin hazırlamaktadır.

---

5. Sonuç ve Öneriler

Bu teknik tasarım analizi, Dynatrace Monitored Entities API v2'nin, büyük ölçekli bir mikroservis ortamında servis topolojisi çıkarma görevini otomatikleştirmek için modern, ölçeklenebilir ve teknik olarak tek doğru çözüm olduğunu teyit etmektedir. Proje, SRE ekibinin operasyonel verimliliğini artırmak, hata analiz süreçlerini hızlandırmak ve CMDB gibi kurumsal sistemlerle entegrasyonu sağlamak için yüksek bir stratejik değere sahiptir ve tamamen uygulanabilirdir.

Projenin başarısını garantilemek ve kurumsal düzeyde sürdürülebilir bir otomasyon aracı oluşturmak için aşağıdaki en iyi uygulama tavsiyelerine kesinlikle uyulması zorunludur:

1. **Verimli ve Hedefe Odaklı Sorgulama** En yüksek verimlilik için her API isteği, `entitySelector=type("SERVICE")` filtresini, veri yükünü en aza indirmek amacıyla `fields=+fromRelationships.calls,+toRelationships.called_by` parametresiyle birleştirmelidir. Bu yaklaşım, gereksiz verilerin transferini engelleyerek ağ trafiğini, API işlem süresini ve istemci tarafındaki ayrıştırma (parsing) yükünü minimize eder.
2. **Ölçeklenebilir ve Hatasız Sayfalama** Binlerce servisin bulunduğu ortamlarda tüm veriyi eksiksiz çekmek için, API çağrı sayısını azaltmak amacıyla

`pageSize=500` kullanılmalıdır. Otomasyon betiği, `nextPageKey` imleç mekanizmasını temel alan durum-tabanlı (stateful) bir döngü mimarisi uygulamalıdır. Sonraki sayfa isteklerinde `entitySelector` ve `fields` gibi diğer sorgu parametrelerini göndermeme kuralına uyulması, `HTTP 400` hatalarını önlemek için zorunludur.

3. **Dayanıklı ve Güvenli Çalışma** Üretim ortamı için geliştirilen betiklere, API limitlerine takılıp kesintiye uğramamak için `X-RateLimit-Reset` HTTP başlığına dayalı senkronize bir geri çekilme (backoff) mekanizması eklenmelidir. Güvenliği en üst düzeye çıkarmak için, En Az Ayrıcalık İlkesi'ne uygun olarak **yalnızca `entities.read` kapsamına sahip API token'ları kullanılmalı** ve bu token'lar kesinlikle kaynak koduna eklenmeden, HashiCorp Vault gibi güvenli bir sır yönetim sistemi üzerinden yönetilmelidir.