Dağıtık Sistem Mimarileri

Bu derste dağıtık sistem mimarilerini ele alacağız. Farklı mimari yaklaşımları teorik olarak inceleyerek uygulama örneklerini görmeye çalışacağız.

1. Monolitik, Mikroservis ve SOA Mimarileri

1.1 Monolitik Mimari

Monolitik mimari, yazılım bileşenlerinin tek bir yapı altında birleştiği, tarihsel olarak en çok kullanılan uygulama mimarilerinden biridir. Bu mimaride kullanıcı arayüzü, iş mantığı ve veri erişim katmanları tek bir kod tabanı içinde geliştirilir ve tek bir dağıtım birimi olarak çalıştırılır. Geliştirilen uygulama, genellikle bir JAR, WAR ya da EXE dosyasına derlenerek çalıştırılır.

Bu yaklaşım, özellikle küçük ve orta ölçekli projelerde geliştirme sürecini hızlandırması bakımından tercih edilir. Ancak zamanla artan sistem karmaşıklığı ve büyüyen ekiplerle birlikte bu mimarinin sınırlamaları belirginleşir.

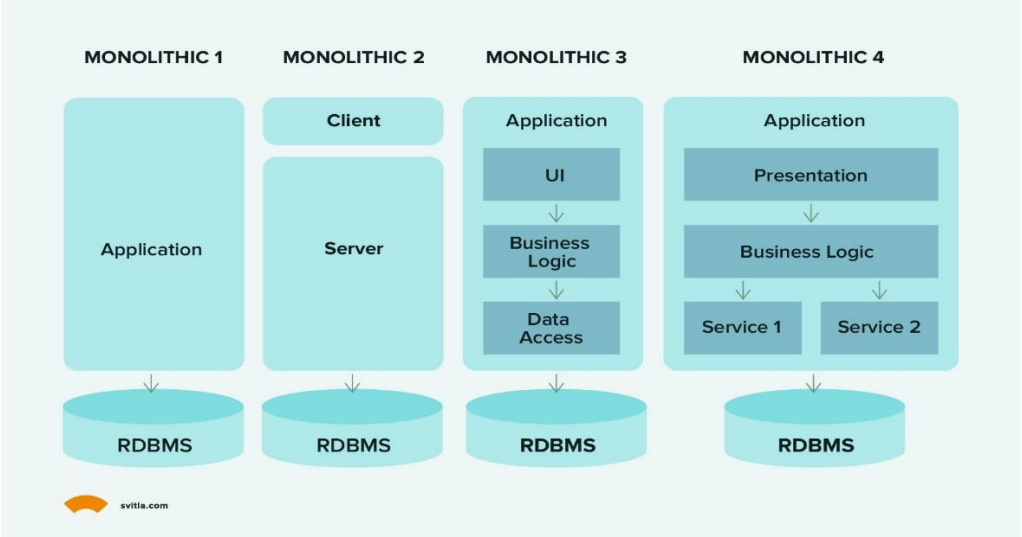
Monolitik Mimarinin Temel Özellikleri

1. **T**ek Kod Tabanı: Tüm bileşenler (veritabanı, iş mantığı, kullanıcı arayüzü) aynı kod tabanı içinde yer alır.
2. Tek Dağıtım Birimi**:** Uygulama tek bir dosya (JAR, WAR, EXE, vb.) olarak dağıtılır.
3. Bağımlı Bileşenler**:** Bir bileşende yapılan değişiklikler diğer bileşenleri etkileyebilir.
4. Kolay Başlangıç**:** Yeni başlayanlar için geliştirmesi ve test edilmesi kolaydır.
5. Zor Ölçeklenme**:** Artan kullanıcı sayısı karşısında yatay ölçeklenmesi (sunucu ekleyerek) zordur.

Monolitik Mimarinin Avantajları ve Dezavantajları

Avantajlar	Dezavantajlar
Kolay geliştirme: Başlangıç için hızlıdır.	Ölçeklenme zorluğu: Tüm uygulamanın aynı anda büyümesi gerekir.
Kolay test edilebilirlik: Tüm sistem tek bir kod tabanında olduğu için test süreçleri daha basittir.	Tek hata noktası (SPOF - Single Point of Failure): Bir bileşenin çökmesi tüm sistemi etkileyebilir.
Basit dağıtım süreci: Tek bir dosya veya sunucuya yüklenir.	Değişiklik zorluğu: Küçük bir değişiklik bile tüm uygulamanın yeniden dağıtılmasını gerektirir.
Performans açısından verimli: İç iletişim süreci daha hızlıdır.	Ekip yönetimi zorluğu: Büyük ekipler aynı kod tabanı üzerinde çalışırken karmaşıklık artar.

Monolitik mimari genellikle 3 temel katmandan oluşur: (i) Kullanıcı Arayüzü (UI) → Web/Mobil arayüz, (ii) İş Mantığı (Business Logic) → İş kurallarını yöneten kodlar, (iii) Veri Tabanı (Database) → Tüm verinin saklandığı merkezi yapı.



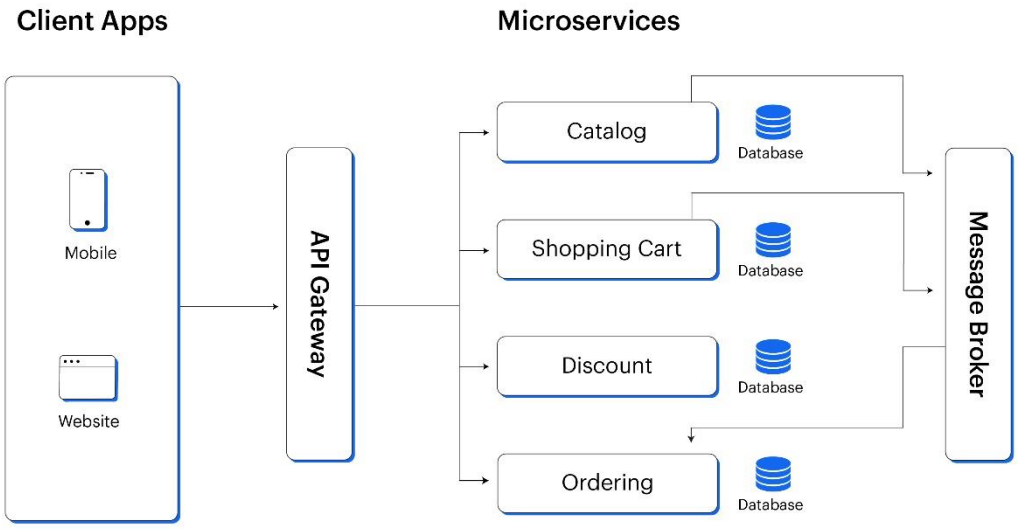
Böyle bir sistemde bir noktada oluşan problem tüm sistemi etkileyebilir. Örneğin; bir e-ticaret sitesinde ürün ekleme, sipariş verme, ödeme yapma gibi işlemler aynı uygulama içinde gerçekleştirilir. Eğer ödeme sistemi çökerse, tüm site etkilenebilir. Ayrıca geliştirme süreçleri de yavaş olabilir, Debug yapıldığında tüm sistemin ayağa kaldırılması gerekecektir.

1.2 Mikroservis (Microservice) Mimarisi

Mikroservis mimarisi, bir uygulamanın bağımsız çalışan küçük servisler halinde geliştirilmesine dayanan dağıtık bir yazılım mimarisidir. Her servis, tek bir işlevi yerine getirir ve kendi veritabanına sahiptir. Mikroservisler, API'ler veya mesajlaşma sistemleri aracılığıyla haberleşir. Örneğin Amazon'un ürün kataloğu, sipariş yönetimi, ödeme, kargo takip servisleri birbirinden bağımsızdır.

Mikroservis Mimarinin Temel Özellikleri

1. Bağımsız Servisler**:** Her servis, belirli bir işlevi yerine getirir. Örneğin, bir e-ticaret sitesinde sipariş yönetimi**,** kullanıcı yönetimi**,** ödeme gibi bileşenler ayrı çalışır.
2. Ölçeklenebilirlik**:** Sadece yoğun kullanılan servisler ölçeklenebilir (örneğin, ödeme sistemi daha fazla sunucuya yayılabilir).
3. Bağımsız Dağıtım**:** Bir serviste yapılan güncelleme, diğerlerini etkilemez.
4. Çoklu Programlama Dili Desteği**:** Her servis, ihtiyaca göre farklı bir dil veya teknolojiyle geliştirilebilir.
5. Veri Bağımsızlığı**:** Her servis, kendi veritabanına sahiptir (DB paylaşımı yerine veri bölme yöntemi - "data partitioning").



Mikroservisler kendi başlarına çalışsa da, genellikle birbirleriyle etkileşim halindedir. Bu iletişim şu şekillerde gerçekleşebilir:

- RESTful API (HTTP üzerinden)
- gRPC (Protobuf tabanlı ikili iletişim)
- Mesajlaşma Sistemleri (Kafka, RabbitMQ, Amazon SQS gibi asenkron yapılandırmalar)

Bu yapı sayesinde yüksek trafik altındaki bileşenlerde asenkronlaştırma sağlanarak sistem genelinde darboğazlar önlenebilir.

API Gateway: Mikroservis sistemlerinde kullanıcıdan gelen isteklerin doğrudan servislere yönlendirilmesi karmaşaya yol açar. Bu nedenle genellikle API Gateway kullanılır. Bu katman, gelen istekleri uygun mikroservislere yönlendirir, doğrulama yapar, rate limit uygular ve hata yönetimini merkezileştirir.

Mikroservis Mimarisinin Avantajları ve Dezavantajları

Avantajlar	Dezavantajlar
Bağımsız geliştirme: Farklı ekipler farklı servisleri geliştirebilir.	Servisler arası iletişim karmaşıktır: API yönetimi ve mesajlaşma zorluk yaratabilir.
Bağımsız ölçeklenme: Sadece yoğun servisler ölçeklenebilir.	Dağıtık veri yönetimi: Veritabanları birbirinden bağımsız olduğu için tutarlılık sorunları yaşanabilir.
Bağımsız dağıtım: Tek bir servis güncellenebilir, tüm sistemi yeniden başlatmaya gerek yoktur.	Daha fazla operasyonel yük: Servisler ayrı ayrı yönetildiği için DevOps süreçleri daha karmaşıktır.
Esneklik (Farklı teknolojiler kullanma imkanı)	Performans sorunları olabilir: Servisler arasındaki fazla istek, ağ trafiğini artırabilir.

Monolitik ve Mikroservis Mimari Karşılaştırması

Özellik	Monolitik Mimari	Mikroservis Mimari
Kod Yapısı	Tüm bileşenler tek bir kod tabanında	Küçük, bağımsız servislerden oluşur
Ölçeklenebilirlik	Zor (bütün sistem büyütülmeli)	Kolay (sadece ihtiyaca göre büyütülür)
Bakım & Güncelleme	Tüm uygulama etkilenir	Servis bazlı güncellemeler mümkün
Bağımlılıklar	Bir değişiklik tüm sistemi etkileyebilir	Servisler birbirinden bağımsız çalışır
Geliştirme Süreci	Tek bir ekip çalışır	Ekipler farklı servislerde çalışabilir
Dağıtım Süreci	Komple dağıtım gerekir	Servisler tek tek güncellenebilir

Ne Zaman Monolitik?	Ne Zaman Mikroservis?
• Küçük projelerde • Tek bir ekip tarafından geliştirilen uygulamalarda	• Büyük ölçekli projelerde • Birden fazla ekibin çalıştığı uygulamalarda • Yüksek ölçeklenebilirlik ihtiyacında

Mikroservisler İçin Kullanılan Teknolojiler

Servis Geliştirme:
- ⇒ Spring Boot (Java)
- ⇒ Node.js (JavaScript)
- ⇒ Django (Python)
- ⇒ Ruby on Rails (Ruby)
- ⇒ .NET Core

API Yönetimi & İletişim:
- ⇒ RESTful API
- ⇒ GraphQL
- ⇒ gRPC
- ⇒ RabbitMQ, Kafka (Asenkron Mesajlaşma)

Veri Yönetimi:
- ⇒ PostgreSQL, MSSQL, Oracle, MySQL (İlişkisel Veritabanları)
- ⇒ MongoDB, Cassandra, Neo4j (NoSQL)
- ⇒ Redis (Önbellekleme)

Konteyner & Dağıtım:

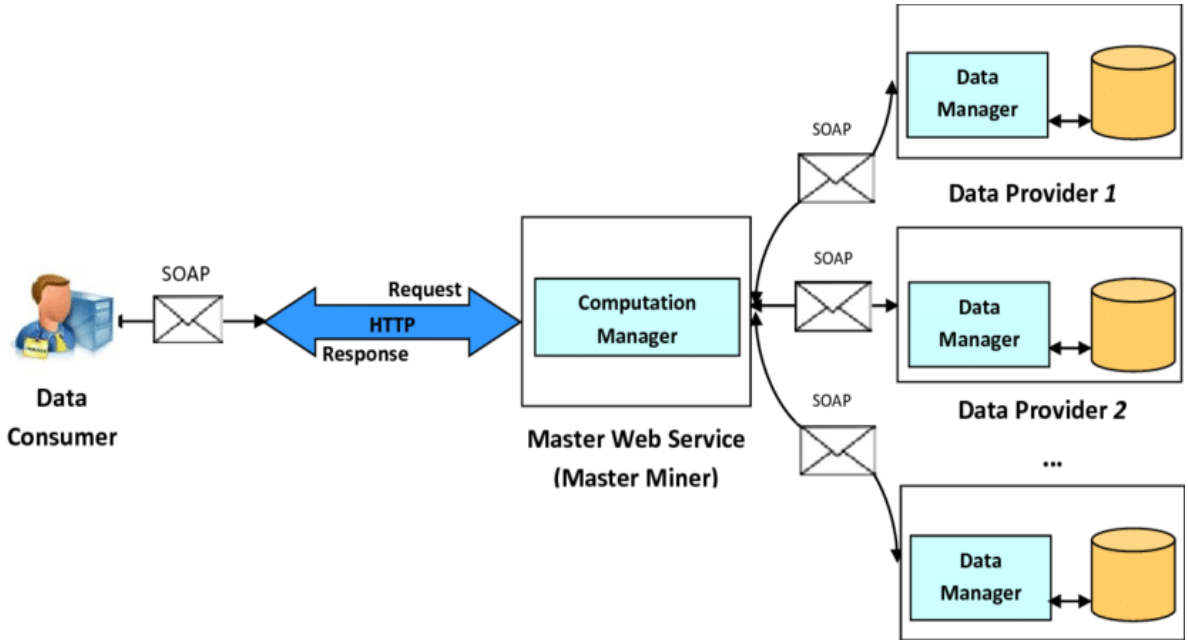
- ⇒ Docker
- ⇒ Kubernetes
- ⇒ AWS Lambda (Serverless Mimari)

1.3 SOA (Service-Oriented Architecture) Mimarisi

SOA (Service-Oriented Architecture - Servis Odaklı Mimari), farklı yazılım bileşenlerinin bağımsız servisler olarak geliştirilip, birbirleriyle standart protokoller (HTTP, SOAP, REST, gRPC) aracılığıyla iletişim kurduğu bir mimari yaklaşımdır. SOA'nın temel amacı, uygulamaları servis bazlı bileşenlere ayırarak esneklik, yeniden kullanılabilirlik ve entegrasyon kolaylığı sağlamaktır.

SOA Mimarisinin Temel Özellikleri

1. Bağımsız Servisler**:** Her servis, belirli bir işlevi yerine getirir ve tek başına çalışabilir.
2. Standart Protokollerle İletişim: Servisler SOAP, REST, XML veya gRPC ile haberleşir.
3. Yeniden Kullanılabilirlik: Aynı servis farklı uygulamalarda kullanılabilir.
4. Platform Bağımsızlığı: Farklı programlama dillerinde geliştirilen servisler birbiriyle entegre olabilir.
5. Gevşek Bağlılık (Loose Coupling): Servisler birbiriyle bağımlı değildir, bir servis değiştiğinde diğerleri etkilenmez.



SOA servisleri Enterprise Service Bus (ESB) veya API Gateway kullanarak birbirleriyle iletişim kurar.

SOA Mimarisi Avantajları ve Dezavantajları

Avantajlar	Dezavantajlar
Yeniden Kullanılabilirlik: Servisler farklı projelerde tekrar kullanılabilir.	Performans Sorunu: Servisler arası iletişim HTTP, XML gibi formatlarla olduğu için ek yük getirebilir.
Kolay Entegrasyon: Farklı sistemler birbiriyle entegre edilebilir.	Karmaşıklık: Çok fazla servis olduğunda yönetim zorlaşır.
Bağımsız Güncelleme: Her servis ayrı güncellenebilir.	Bağlantı Yönetimi Zordur: Servisler arası çağrılar karmaşık hale gelebilir.
Farklı Teknolojilerle Çalışabilir	Geliştirme ve Test Süreçleri Daha Uzundur

SOA ve Mikroservis Mimarisi Karşılaştırması

Özellik	SOA (Service-Oriented Architecture)	Mikroservis Mimari
Servis Yapısı	Büyük, bağımsız servislerden oluşur	Küçük, bağımsız mikroservislerden oluşur
İletişim Protokolü	SOAP, REST, XML	REST, gRPC, Event-Driven
Bağımlılık	Servisler gevşek bağlıdır ancak entegrasyon daha yoğundur	Servisler tamamen bağımsızdır
Ölçeklenebilirlik	Daha az esnektir	Daha esnek ve dinamik
Entegrasyon	Mevcut sistemlere kolay entegre edilebilir	Mikroservislerin uyum sağlaması gerekir
Kullanım Alanı	Büyük ölçekli kurumsal sistemler	Modern, bulut tabanlı uygulamalar

Ne Zaman SOA Kullanılmalı?	**Ne Zaman Mikroservis Kullanılmalı?**
Büyük kurumsal sistemlerde (ERP, CRM) Farklı teknolojilerle entegrasyon gerektiren uygulamalarda Merkezi servis yönetimi ihtiyacı varsa	Yüksek ölçeklenebilirlik gerektiren sistemlerde Çevik (Agile) geliştirme yaklaşımı benimsendiğinde Hafif, bağımsız servisler geliştirilmek istendiğinde