MONOLITHIC MİMARİSİ

*“Monolithic architecture yazılımın **self-contained**(kendi kendine yeten) olarak tasarlanması anlamına gelmektedir. Bir standart doğrultusunda “tek bir parça” olarak oluşması da diyebiliriz. Bu mimarideki component’ler **loosely coupled** olmasından ziyade, **interdependent** olarak tasarlanmaktadır.”*

Monolithic mimari kendi kendine yetebilecek bir uygulamada ki bütün fonksiyonalitelerin tek bir çatı altında geliştirilmesidir.

AVANTAJLARI

1. Yönetilebilirlik ve monitoring kolaydır.
2. Küçük çaplı projeler için geliştirilmesi ve bakımı kolaydır. Hızlı bir şekilde uygulama geliştirilebilir.
3. Fonksiyonalitelerin birlikte çalışabilirliği tutarlıdır.
4. Transaction yönetimi kolaydır.

DEZAVANTAJLARI

1. Uygulama büyüdükçe yeni özellik geliştirilmesi ve mevcut kodun bakımı zorlaşır.
2. Projede çalışan ekip sayısının ve çalışan sayısının artması ile geliştirme ve bakım daha güç hale gelir.
3. Birbirlerine olan bağımlılıklarından dolayı, bir fonksiyonalitede yapılan değişiklik diğer yerleri etkileyebilir.
4. Spesifik bir fonksiyonaliteyi scale edebilme imkanı yoktur. (Örneğin geliştirdiğiniz uygulamada sürekli fatura oluşturuluyor ve burası uygulamanın dar boğazı. Siz bu fonksiyonaliteyi birden fazla instance da çalıştırmak isteseniz bile uygulamanız monolithic mimaride olduğu için sadece ilgili servis yerine bütün uygulamayı scale etmek zorunda kalırsınız.)
5. Versiyon yönetimi zorlaşır.
6. Uygulamada aynı programlama dili ve aynı frameworklerin kullanılması gerekir.
7. Uygulamada yapılan küçük bir değişiklikte bile bütün uygulamanın deploy olması gerekir.

MICROSERVICE MİMARİSİ

Microservice tam olarak şöyle tanımlanabilir: Birbirinden bağımsız olarak çalışan ve birbirleriyle haberleşen bireysel servislerdir. Her servis kendisine ait olan iş mantığını yürütür ve diğer servislerin iş mantığı ile ilgilenmez. Plansız bir şekilde genişleyen monolithic uygulamanın hantallığını, karmaşıklığını azaltan ve yönetimini kolaylaştıran bir mimaridir.

Servisler arasında ki iletişim HTTP, AMQP, TCP, UDP vb... gibi protokoller ile gerçekleşir. Transfer edilecek verilerin niteliğine göre uygun protokol seçilir. HTTP protokolü ile servisler arasında veriler JSON, XML veya Binary formatıyla gönderilir.

AVANTAJLARI

1. Uygulama çok büyük de olsa çok küçük de olsa yeni özellik eklenmesi ve mevcut kodun bakımı kolaydır. Çünkü sadece ilgili servis içerisinde değişiklik yapmak yeterlidir.
2. Her servis birbirinden bağımsız ve sadece kendi iş mantıklarını bulundurduğu için servisin code base'i oldukça sade olacaktır.
3. Ekipler daha verimli ve hızlı bir şekilde çalışabilir. Projeye yeni başlayan arkadaşlar code base de kaybolmadan kolay bir şekilde adapte olabilir.
4. Servisler birbirinden bağımsız bir şekilde scale edilebilir.
5. Versiyonlama kolay bir şekilde gerçekleştirilir.
6. Bir servisde yapılan değişiklik, diğer servislerin deploy yapılmasını gerektirmez. Sadece ilgili servisin deploy yapılması yeterlidir.
7. Servisler farklı dillerde ve farklı frameworkler ile yazılabilir. Her servisin kendine ait farklı veritabanı olabilir.

DEZAVANTAJLARI

1. Birden fazla servis ve birden fazla veritabanı olduğu için transaction yönetimi zorlaşacaktır.
2. Bu servislerin yönetilebilirliği ve monitoring işlemi zorlaşacaktır.
3. Servisler gereğinden fazla bağımsız olursa yönetimi zorlaşacaktır. Örneğin **user** ile alaklı **CRUD** işlemler yapan **user-service** adında bir servisiniz olduğunu varsayalım. Eğer siz gaza gelip **user-service**'ın her bir fonksiyonalitesini (create, update, delete, get... vb) ayrı servislere ayırmak isterseniz bunun yönetimi çok daha zorlaşacaktır.