# 1. Conway’in Kanunu’ nu (Conway’s Law) açıklayınız.

Sistem tasarlayan kuruluşlar,bu kuruluşların iletişim yapılarının kopyaları olan tasarımlar üretmekle sınırlıdır.Conway law olarak bilinen bu ifade deneysel olarak kanıtlanmış ve titiz çalışmalarda gösterilmiştir.Conway yasasının pratik anlamı, organizasyonlarımızı “düşünme şeklimizin” geliştirdiğimiz sistemler üzerinde doğrudan bir etkiye sahip olmasıdır.Monolitik olarak organize edilmiş bir ekiple bir organizasyonda microserviceler tasarlamak mümkün değildir.Loosely coupled ve verimli componentler istiyorsak, ekipler bu şekilde çalışan ekipler olmalıdır.Özerk ancak hizalanmamış ekipler, birlikte iyi çalışmayan loosely coupled componentlar üretirler.Bir sistemdeki iki component çakışıyorsa organizasyonda çakışan iki takım vardır.

# 2. Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), On-Premises kavramlarını örneklerle açıklayınız.

**SaaS**

SaaS kısaca; müşteriler tarafından ulaşılmak istenen verilere internet bağlantısı ve web tarayıcısı olan herhangi bir cihazdan erişilmesini sağlayan bulut tabanlı bir yazılım sağlama servisidir. Bu servis ile yazılımı sağlayıcısı, sunucu ve veritabanı gibi sistemleri barındırmak ve bunların bakımını yapmakla sorumludur.

**PaaS**

Platform as a Service (PaaS) yani Hizmet Olarak Platform, üçüncü taraf bir bulut hizmet sağlayıcısının, genellikle uygulama barındırma veya geliştirme için gerekli olan bazı donanım ve yazılım araçlarını, müşterilere internet üzerinden sunduğu bir bulut hizmeti sunumu modelidir. Bu platform; bilgi işlem, bellek, depolama, veri tabanı ve diğer uygulama geliştirme hizmetlerini içerir. PaaS çözümleri, dahili kullanım için yazılım geliştirmek için kullanılabilir veya satışa sunulabilir.

**IaaS**

IaaS modelinde sağlayıcınız; sunucular, depolama, ağ donanımı ve sanallaştırma (hipervizör) katmanı dahil olmak üzere uygulama ve hizmetler sunmak için ihtiyaç duyduğunuz tüm fiziksel ve sanal çekirdek altyapıyı sağlar ve barındırır. Bileşenler çalışmaya başladığında DevOps görevi devralır ve gerekli tüm ara katman yazılımlarını, uygulama yazılımlarını, veri tabanlarını vb. yükler ve bakımını yapar. Sağlayıcınıza kullandığınız kaynaklar için abonelik temelinde ödeme yapar ve tüketiminizi ölçeklendirebilirsiniz.

**On-Premises**

Şirket içi yazılım, sunucu grubu veya bulut gibi uzak bir tesiste değil, yazılımı kullanan kişi veya kuruluşun tesislerinde bulunan bilgisayarlara kurulur ve çalıştırılır. Şirket içi yazılıma bazen "shrinkwrap" yazılımı denir ve şirket dışı yazılıma genellikle "hizmet olarak yazılım" veya "bulut bilişim" denir.

# 3. Continuous Integration, Continuous Delivery ve Continuous Deployment kavramlarını açıklayınız.

CI, Sürekli entegrasyon, kod üzerinde yapılan değişiklik sonrasında sistemin çalışır durumda olduğunu, yapılan değişikliğin sorunlara yol açmadığını tespit etmek için kullanılan yöntemdir. Sorunları ve kırılmaları tespit edebilmek için birim testleri kullanılır. Yapılan değişiklikler yeni bir yapının parçası olduğundan dolayı testlerde oluşan hatalar, yapılan değişikliğin sistemi kırdığı anlamına gelmektedir. Bu durumda tüm programcılar bilgilendirilerek hatanın bir an önce giderilmesi sağlanır. Sürekli entegrasyon ile programcılar tarafından kod üzerinde yapılan çalışmalar neticesinde her zaman çalışır bir sürümün oluşması sağlanmış olur.

CD, Sürekli teslimat, sürekli entegrasyonun doğal bir uzantısıdır.

Sürekli teslimat, başarılı olan bir yapıyı (build) bir ortama atma durumunun otomatik olan yoludur. Teslimat ile dağıtım arasında küçük bir fark vardır. Teslimat manuel yolla, dağıtım ise otomatik yapılır. Sürekli teslimat düzgün bir şekilde uygulanırsa, müşteriler standartlaştırılmış bir test sürecinden geçmiş yapıya sahip olacaklardır.

Örnek verecek olursak; Netflix, tamamen otomatik bir sürekli dağıtım sistemine sahiptir. Dağıtımlar otomatik olduğundan ve herhangi bir zamanda gerçekleşebileceğinden, uygulamanız, yeni bir sürüm dağıtıldığında geçici kesintilerden etkilenmeyecek şekilde tasarlanmalıdır.

# 4. API Gateway pattern’ı açıklayınız.

# API Gateway pattern'ı birden çok istemci uygulamasıyla karmaşık, büyük mikro hizmet tabanlı uygulamalar tasarlamak ve oluşturmak istiyorsanız önerilir. Model, nesne yönelimli tasarımdaki cephe modeline benzer, ancak senkronize bir iletişim modeli olarak kullanmak için dağıtılmış bir sistem ters proxy veya ağ geçidi yönlendirmesinin bir parçasıdır.

# 5. Backend for frontend (BFF) pattern’ ı açıklayınız.

Her client için ayrı bir api-gateway tanımlanmasını konu alır. Yani yukarıdaki 3 durum için (web-ui,mobile-app,3rd app) 3 farklı BFF ile sorunlarımızı çözümlemeye çalışan pattern’e denir.

**BFF’in avantajları;**

Olası problemlerin yönetimini kolaylaştırır.

Her client için mikroservisleri bölümlere ayırır.

Business logic’i mikroservislerden alarak tek bir nokta üzerine almamızı sağlar.

**BFF’in dezavantajları;**

Uygulama sayısının artmasıyla birlikte bakım ve operasyon maliyetlerinin artması

Mimari karmaşıklığın artması

Eklenecek ayrı bir katmanın ağ isteklerinde gecikme yaratma ihtimali

BFF’lerde kod tekrarlarının yapılma ihtimali

# 6. Circuit-breaker pattern’ ı açıklayınız.

Circuit Breaker Pattern, bir yazılımda hataları tespit ederek hatanın tekrar etmemesini sağlar. Böylece sistemde hataların tekrar etmesi sonucu oluşacak bloklanma, hizmet kesintileri, aşırı kaynak kullanımı gibi sorunlarla karşılaşılmasını önleyerek, sistemin sağlıklı çalışmasını amaçlar.

# 7. Microservice chassis pattern’ ı kısaca açıklayınız

Microservice chassis pattern temelde işlevsel mantığı tek bir modülde birleştirmeyi ve onu çeşitli servicelerde yeniden kullanım için hazırlamayı amaçlar.Aopnin dışa dönük uygulama özelliklerini ayrı kod birimlerine ayırması gibi,chassis modeli de microservice mimarilerini çoğaltan back end işlevleri için ayrı modüller oluşturur.Microservice chassisin bazı avantajları vardır.

-Cross cutting concernleri ele almak için developerların yazması gereken kod miktarını azaltır.

-Business logic bir uygulamanın işlevsel kaygılarından ayırmaya yardımcı olur.

-Servicelerin standartlaştırılmış bir dizi yönergeyi izlemesini sağlayarak tutarlılığı arttırır.

Microservice chassis için dikkat edilmesi gereken bazı özellikler vardır.

-Fazlalık olup olmadığına dikkatlice bakarak mekanizmanın kodunu mümkün olduğunca hafif olmalıdır.

-Gereksiz özellikler ayrı bir library arşivine taşınmalıdır.

-Microservice mimarisi birden çok yazılım dili içeriyorsa, microservice chassisin yazılım dilinden bağımsız yapısını azaltmak için sidecar Proxy kullanılmalıdır.

-Temel çerçeve önemli ölçüde değişmediyse, chassisyi tekrar tekrar test etmekten kaçınılmalıdır.