1. Conway'in Kanunu' nu (Conway's Law) açıklayınız.

Yazılım üretmek karmaşık bir sorumluluktur. Birçok insanın bir arada çalışması gerekebilir. Aynı zamanda yazılım üretilirken birçok farklı yol izlenebilir. Bu nedenle organizasyondaki yazılımcılar arasında güçlü, temiz bir iletişim bulunması gerekir. Ekip liderlerinin, yöneticilerin takımdan beklentileri anlamlı olmalıdır, takımdaki yazılımcıların da günlük olarak kendilerinden ne beklendiğini anlamaları gereklidir.

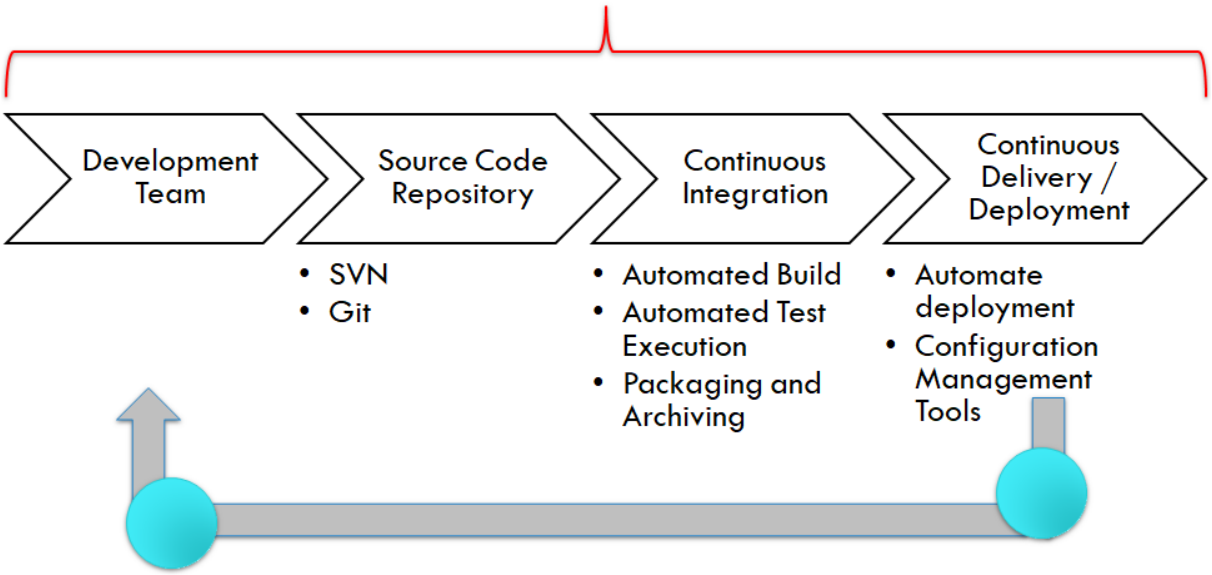
2. Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), On-Premises kavramlarını örneklerle açıklayınız.

SaaS: bulut tabanlı hizmetler, depolama, ağ oluşturma ve sanallaştırma gibi hizmetler sağlar. Kullanıcılar, bu hizmetleri cihazlarına indirmeden veya yüklemeden internet üzerinden bu uygulamalara erişebilirler. Bu hizmetleri kullandıkça ödersiniz veya bir paket satın alırsınız. Örneğin Dropbox servisi size 2GB ücretsiz bulutta depolama alanı verir, daha fazla alan isterseniz buluttan alan satın almanız gerekmektedir.

PaaS: İnternet üzerinden kullanılabilen donanım ve yazılım araçları kullanılabilir. Yazılım ürününün çalışması için bir ortam sağlar. Bu ortamda uygulamalarını oluşturabilir, özelleştirebilir, test edebilirler. Kullanıcılar platformun sağladığı bulut çözümlerini kendi ağlarına dahil edebilirler. Örneğin MySQL veri tabanı isterseniz buluttan bunu satın alıp, kendi ağınıza ekleyebilirsiniz.

IaaS: Sanal Makineler, Depolama (Sabit Diskler), Sunucular, Ağ, Yük Dengeleyiciler vb. çözümler sağlanabilir. Bu hizmette bulutta sağlanan yüksek kapasiteli bilgisayarlar kullanılarak yazılımlar geliştirilebilir.

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



Continuous Integration:

Yaptığımız yeni geliştirmelerin kendi belirlediğimiz koşul doğrultusunda (master'a merge geldiğinde, pull request açıldığında, manuel tetikleme, vs.) bir takım kontrollerden geçtiği ve paketin düzgün derlenip derlenmediğinin de kontrol edildiği süreçtir. Örneğin kod yazarken test'lerini de yazıyorsanız, bu süreç içerisinde mevcut ve yeni eklenen testler ile beraber projenin kontrol edilmesini sağlayabilirsiniz. Code coverage(yazılan testlerin, projenin yüzde kaçını kontrol ettiği) konusunda dikkat ettiğiniz bir projeyse, kapsamın belli bir yüzdenin altına düştüğü durumlarda onay vermeyebilir.

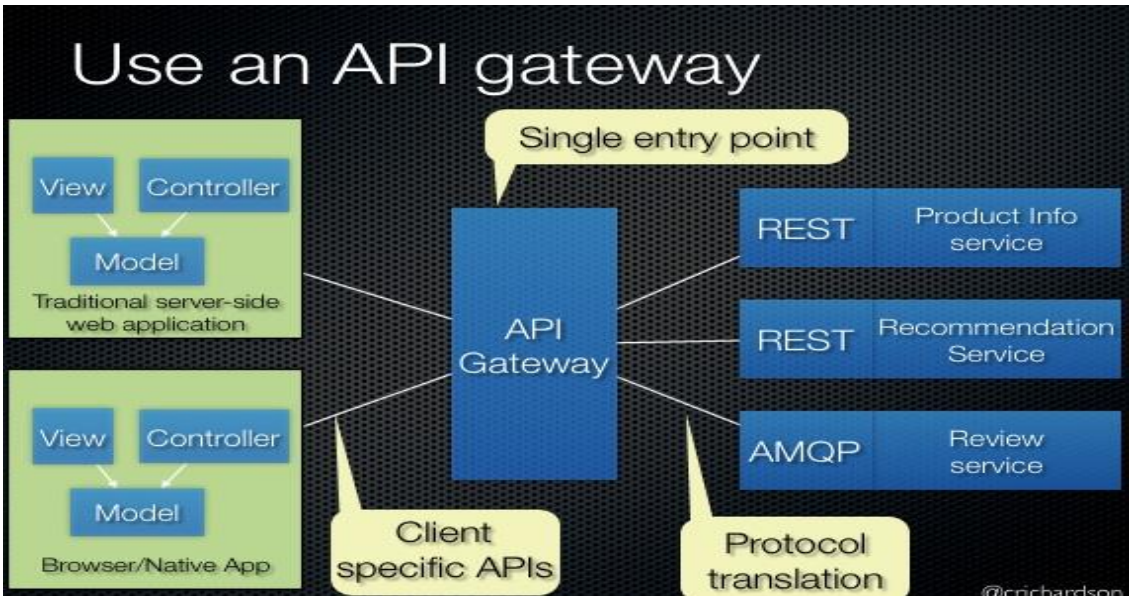
Continuous Delivery; yeni geliştirmeler yapılan yeni kod paketinin CI sürecinden geçtikten sonra build alınıp deploya hazır hale gelmesini sağlayan aşamadır.

Continuous Deployment; hazır hale gelmiş yeni paketi sunuculara kurulmasını sağlayan aşamadır. Bazı firmalarda bu süreç manuel olarak yapılabildiği gibi bazı firmalarda ise bu süreç de otonom bir hale gelmiştir ve CI sürecinden geçen kod doğrudan production ortamına kurulumu yapılabilmektedir.

4. API Gateway pattern'ı açıklayınız.

Tüm istemciler için tek giriş noktası olan bir API Gateway uygulanır. API Gateway, istekleri iki yoldan biriyle işler. Bazı istekler basitçe uygun hizmete yönlendirilir/yönlendirilir. Diğer istekler birçok farklı servislere yönlendirilerek halledilir. API Gateway ile yapılabilecek diğer işlemler :

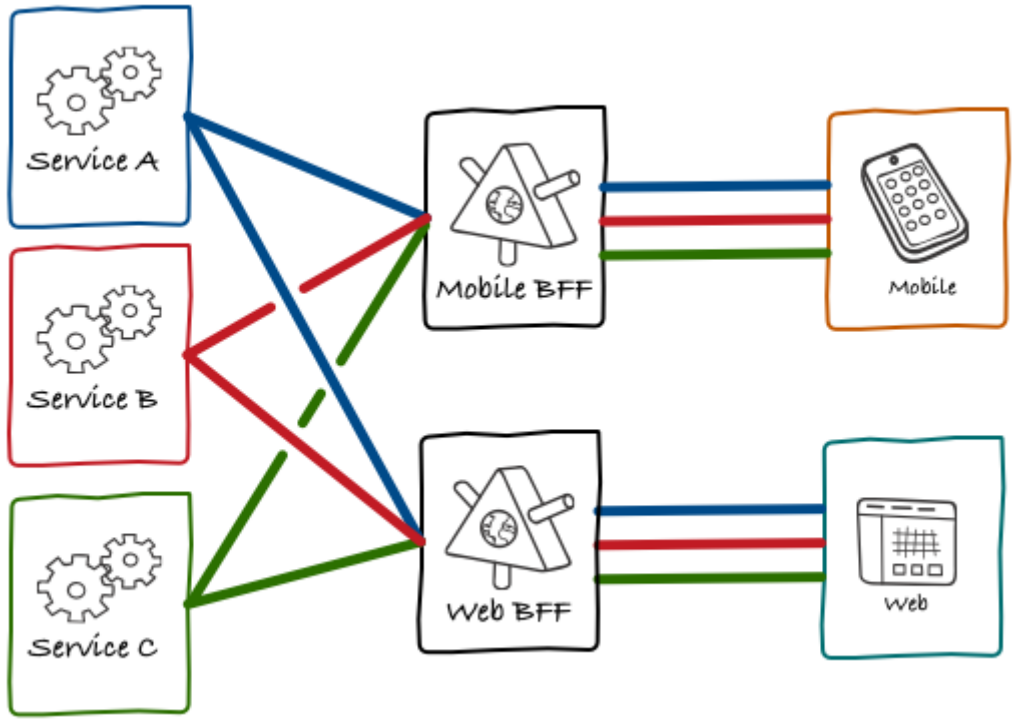
- **Authentication – Authorization :** Servislerimize tek giriş noktası api gatewayimiz olacağından bu servislerin authentication ve authorizationunu api gateway üzerinden yapabiliriz.
- **Logging :** Api gateway üzerinden tüm servislerimize gelen request ve responseları kimin caller'ına kadar loglayabiliriz.
- **Response Caching :** API gateway üzerinden enpointlerinizin responselarını cachleyebilirsiniz. Böylece, endpointe giden call sayısını düşürebiliriz.
- **Routing :** API gateway ile clienti ilgili routelara yönlendirebiliriz



5. Backend for frontend (BFF) pattern' ı açıklayınız.

Bir projede birden fazla client olabilri. Örneğin mobil ve web app gibi. Bu clientler, mikroservise erişirken farklı datalar bekleyebilirler. Yani web app için bir ürünün yorumları, ürünün detayları ile gösterilirken, webb app için ürün yorumları ürünün detayları ile gösterilmeyebilir.

Bu sorunun çözümü için BFF kulanılabilir. Her client için ayrı bir api-gateway tanımlanmasını konu alır.
Her client için ayrı bir BFF Api oluşturarak sorunlara çözüm getirilir.



BFF'in avantajları
Her client için uygun api'yi sağlar.
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. Bu hataların tekrar etmemesini sağlayarak, sistemin yanıt verme süresini kısaltır, aşırı kaynak tüketimini engeller.

Hizmet alınan(istek atılan) bir servisin kullanılabilirliğini kontrol edip oluşabilecek hatalarda sistemimizi belirttiğimiz türdeki sorunlardan korumak için kullanılabilir. Hizmet alınan yapı bir servis, bir veritabanı sunucusu veya uygulama tarafından kullanılan bir web hizmeti olabilir. Circuit breaker, hataları algılar ve uygulamanın başarısız işlem sayısı belirli bir eşiği geçmesi durumunda ise yeniden kullanılabilir hale gelene kadar işlemleri engeller.

7. Microservice chassis pattern' ı kısaca açıklayınız.

Bir mikroservis uygulamasında onlarca hizmet olabilir. Bu mikroservislere ek olarak sürekli yeni servisler geliştirme ihtiyacı olabilir. Bakıma hazır, üretimi yapılmış bir kod tabanın olması bu yazılım sürecini hızlandırabilir, test edilebilirliği kolaylaştırabilir.

Bu hizmetlerin belirli bir şablonlarının olması uygulama geliştirmeyi hızlandıracaktır. Bununla birlikte, bir Hizmet Şablonunun dezavantajı, bir kopyala/yapıştır programlama biçimi olmasıdır: oluşturma mantığı ve kesişme söz konusu olduğunda mantığın değişmesi gerektiğinde, her hizmet ayrı ayrı güncellenmelidir.