DEVOPS

Şirketlerde yazılımın nasıl azıldığını ve hangi süreçlerden geçtiğini biliyor musunuz?

1.Data Center

Şirketlerde yazılan programlar sadece kulanıcıların ulaşabildiği ve güvenli bir ortamda olan ve çalışan bilgisayarlara veya sunuculara bağlanarak gerçekleştirirler. Bu serverler büyük şirketlerde data center yani veri merkezi dediğimiz büyük sayıda bilgi işleme ekipmanını barındıran
büyük sayıda veri işleme ekipmamaını barındıran ve işleten bir tesis veya alandır.

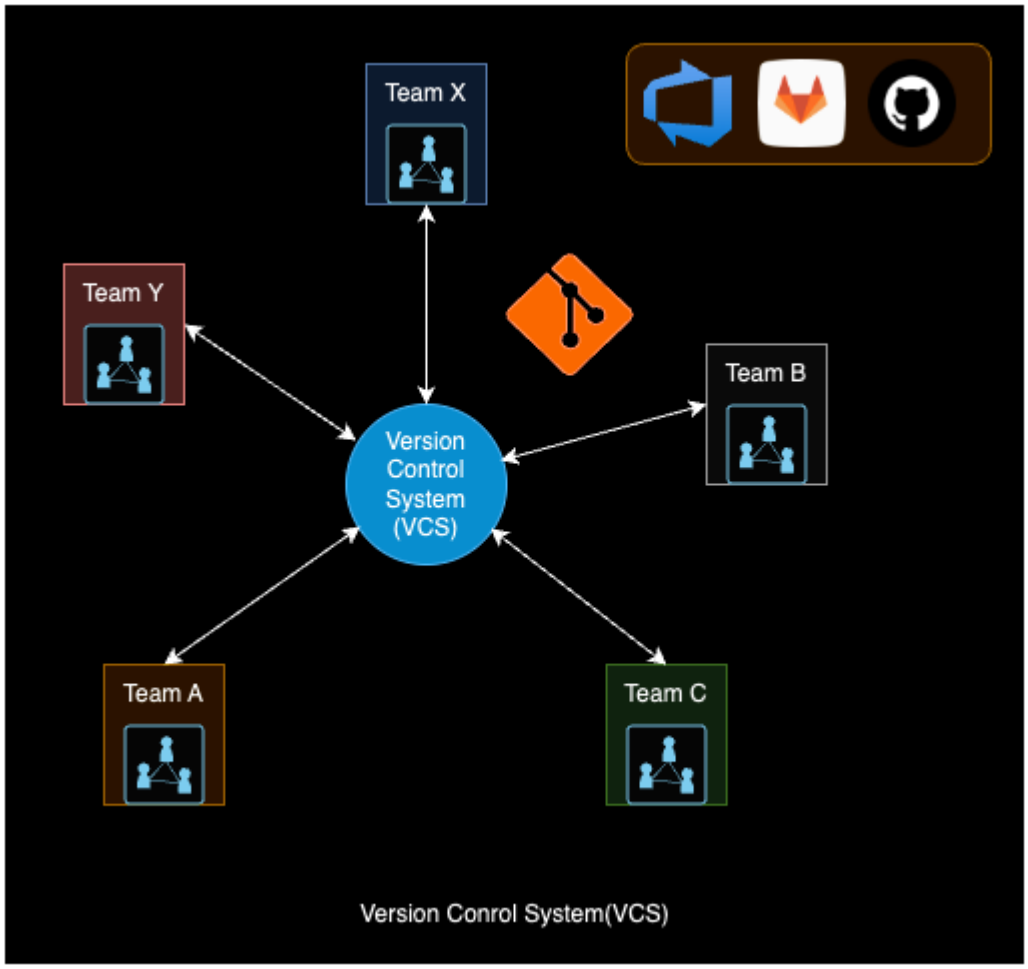
On Premise: Bir kuruluşun kendi tesislerinde, ofis veya veri merkezinde barındırılan ve yönetilen alt yapıdır.

Cloud: Bulut altyapısı internet üzerinden sağlanan genellikle üçüncü taraf bir hizmet sağlayıcı tarafından yönetilen bilişim, kaynaklarını ifade eder. **Azure, Google Cloud, Amazon vb.**

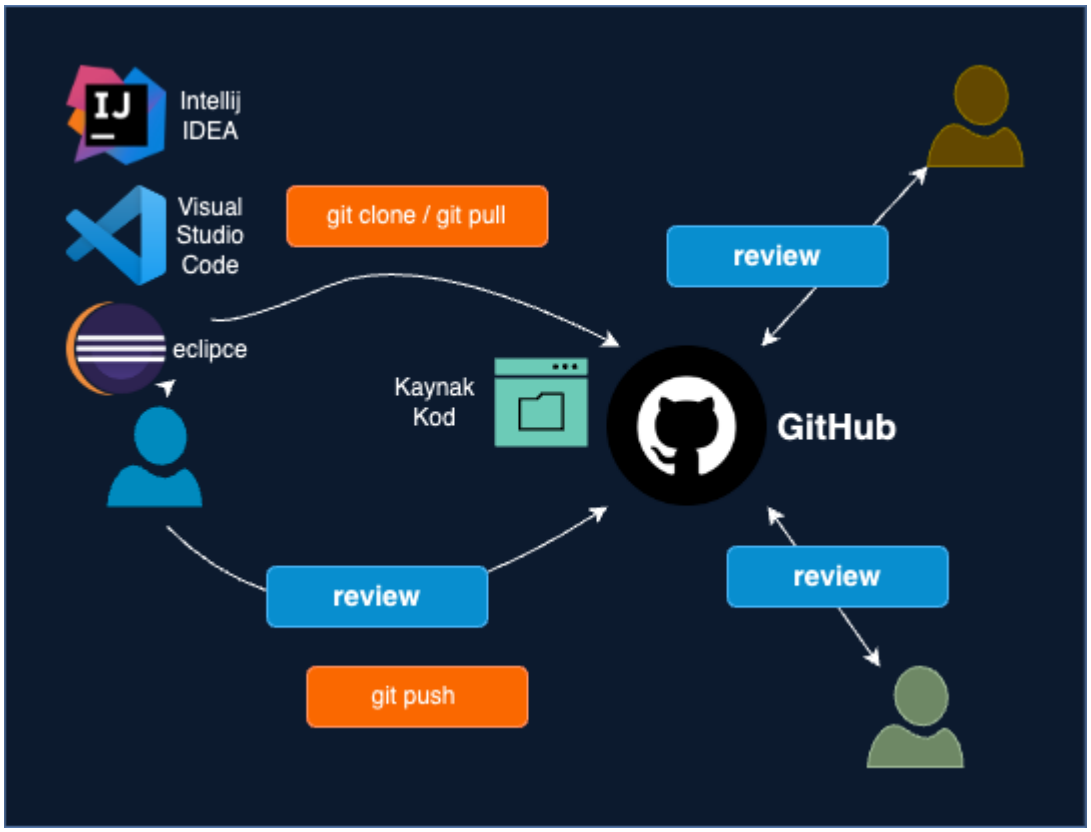
Hybrid: Hem yerinde **on promise** hemde bulutta **cloud** tarafındaki kaynakları birleştiren bir modeldir.

2. Yazılım süreçleri

Şirketlerde yazılım tek başına yazılmaz her zamn takımlar vardır. Küçük işletmelerde istisnalar olabilir. Her takım sorumlu olduğu programları ve kaynak kodlarını devamlı olarak değiştirir ve yönetir. Kaynak kodları kişilerin bilgisayrında kalmayacağından ve birçok kişinin kaynak kodlarında çalıştığından dolayı değişiklikleri takip eden yöneten ve işbirliği yapmak için kullanılan Merkezi kaynak yönetme sistemlerine ihtiyaç duyulur. Bu sistemlere verilen genel isim `Version Control System{:html}` yani sürüm konrotrol sistemleridir. Tüm bu sistemlerin bunları yapmasını sağlayan temel sistemin ismi `GIT{:html}` Her yazılımcı bilgisayarına git kurar ve merkezi çalışan **github** veya **gitlab** gibi sistemlere bağlanıp kaynak kodları ile çalışır.

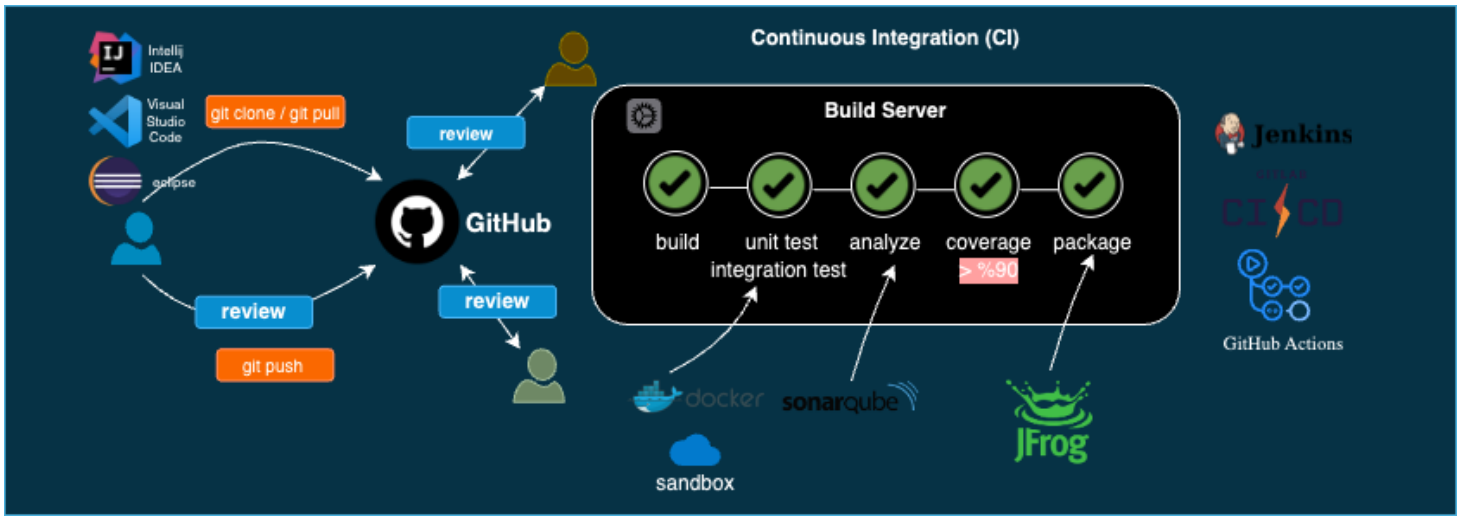


Yazılımcı Kaynak Kodlarına Nasıl Ulaşır ve Nasıl Geri Gönderir?



Öncelikle yazılımcının kaynak kodlarını **github** dan alması gerekiyor. Eğer daha öncesinde kaynak kodunu almamış ise **git clone** komutu ile dosyanın kaynak kodunu kullanır. Dosya zaten var ama değişiklikler almak için **git pull** komutunu kullanır.
Daha sonra yazılım geliştirme ortamlarından(IDE) herhangi biri ile çalışmasını gerçekleştirir. Kodlama tamamladıktan sonra **github** göndermek için **git push** komutunu kullanır. Böylelikle değişiklikleri merkezi sisteme gider. Bu durumda değişikliklerini başkaları da görebilir. Yazılımcı programı gerçekleştirirken tabiki yalnız değildir. Takımda kendi gibi bir çok yazılımcıda vardır. Hepside aynı kaynak kodlarında çalışmaktadır. Takım içinde yazılan kodların merkezi sisteme göndermeden önce gözden geçirilmesi gerekiyor. Bu işlem hataların tespiti kod kalitesinin artırılması ve en iyi ugulama standarlarının uygulanması için önemli bir adımdır.
İnceleme sonucunda takım arkadaşlarınız kodunuzu ya onaylaya bilir ya da istenilen değişiklikleri yapman için sana geri gönderebilir. Merkezi sisteme giriş yapan her kod build etme ve test etme gibi bir çok kontrolden geçer.Bu kontrolleri sağlayan sisteme verilen genel isim ise `buid server{:html}` dır.

2.1 Build



Her değişiklik **build** edilir. Build etmek demek kodu çalıştırılablir br program veya uygulama haline getirmeyi ifade eder. Bu süreç kodun derlenmesi, bağımlılıkların çözülmesi va çalıştırılabilir bir dosya oluşturulmasını içerir.

2.2 Unit test, integration test

Bu adım yazılımın belirli işlevlerinin veya modüllerinin doğru çalıştığından emin olmayı amaçlar, aynı zamanda **integration testlerde** çalıştırılabilir. Bunlar yazılımın kullandığı dış gereksinimler ile beraber nasıl çalıştığını kontol etmek içindir. Servisler arası etkileşimini örnek olarak gösterebiliriz.
Rezervasyon sistemine sahip bir sistemin rezervasyon başarılı olması durumunda rezervasyonApi nin

message servisini çağırmasını örnek olarak verebiliriz.
Bu işlevler kontrol ederken **docker**, **sanbox** gibi uygulamalardan yararlanılabilir.

2.3 Analyze

Kodun analiz edilmesi anlamına gelir. Bu analiz yazılım projelerinde potansiyel hataları, kötü programlama alışkanlıklarını ve güvenlik açıklarını tespit etmeye yöneiktir.Bunu için kullanılan en çok araçlardan biri **sonar** dır.

Analiz ile beraber gelen diğer adım **test coverage**

2.4 Coverage

Bir yazılım projesinde hangi kod bloklarınıın test edildiğini ve hangilerinin test edilmediğinin belirlemek için kullanılır. Genellikle bir yüzde değeri olarak ifade edilir. Bu değer test edilen kodun toplam kod içindeki oranını gösterir. Örneğin 80% test coverage değeri kodun 80% kontrol edildiğii anlamına gelir. **Yazılım projelerinde genelde 85% veya 90% gibi yüksek test coverage kuralı kullanılır.** Bu şu demektir yazılıma giren kodun 90% dan fazlasının kontrol edilmesi gerekiyor. Aksi takdirde hata verir ve işlem sona erer.

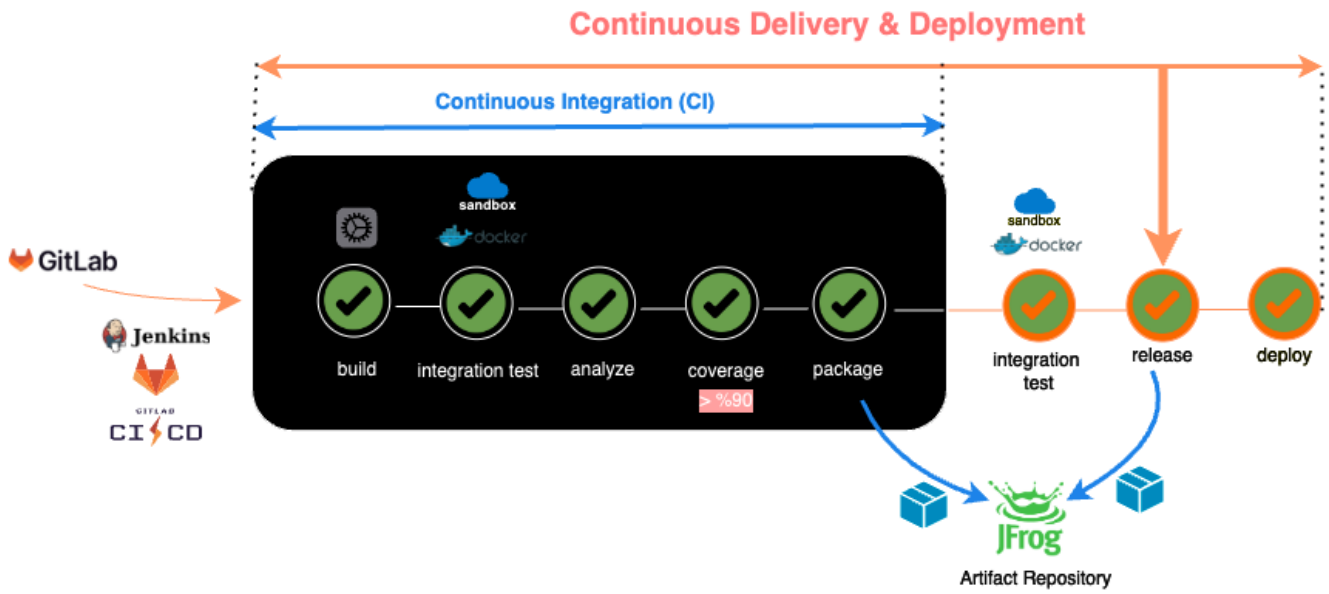
Eğer bütün adımlar doğru çalışmışsa **package** aşamasına yani paketlemeye geçilir.

2.5 Package

Derlenmiş kodun ve diğer gerekli dosyaların geçici bir paket halinde bir araya getirilmesi anlamındadır. Bu paket, yazılımın bir ortamda çalışabilmesi için gereken tüm dosyaları barındırmaktadır. Paketlenen kod bir isim ve versiyon numarası verilerek şirket içinde kullanılan **Artifact Repository** sistemine gönderilir. Burası tüm projelerde kullanılan kütüphaneler, yardımcı dosyalar ve hazırlanmış paketlerin barındığı bir sistemdir. Yani projenizin çalıştırma paketini almak için her defasında tekrar kodu derlemezsiniz. Bu repository içinden alırsınız hazırlanmış paketleri. En çok kullanılan araçlardan biri JFrog Artifactory dir.
Burada anlattığım tüm bu sürecin son adımlarının sırası değişebilir ama her zaman build ile başlar ve package ile sonlanır. İşte bu pratiğe verilen isim
`continuous integration{:html}` dır, kısaltılmışı CI, devamlı entegrasyon anlamındadır.

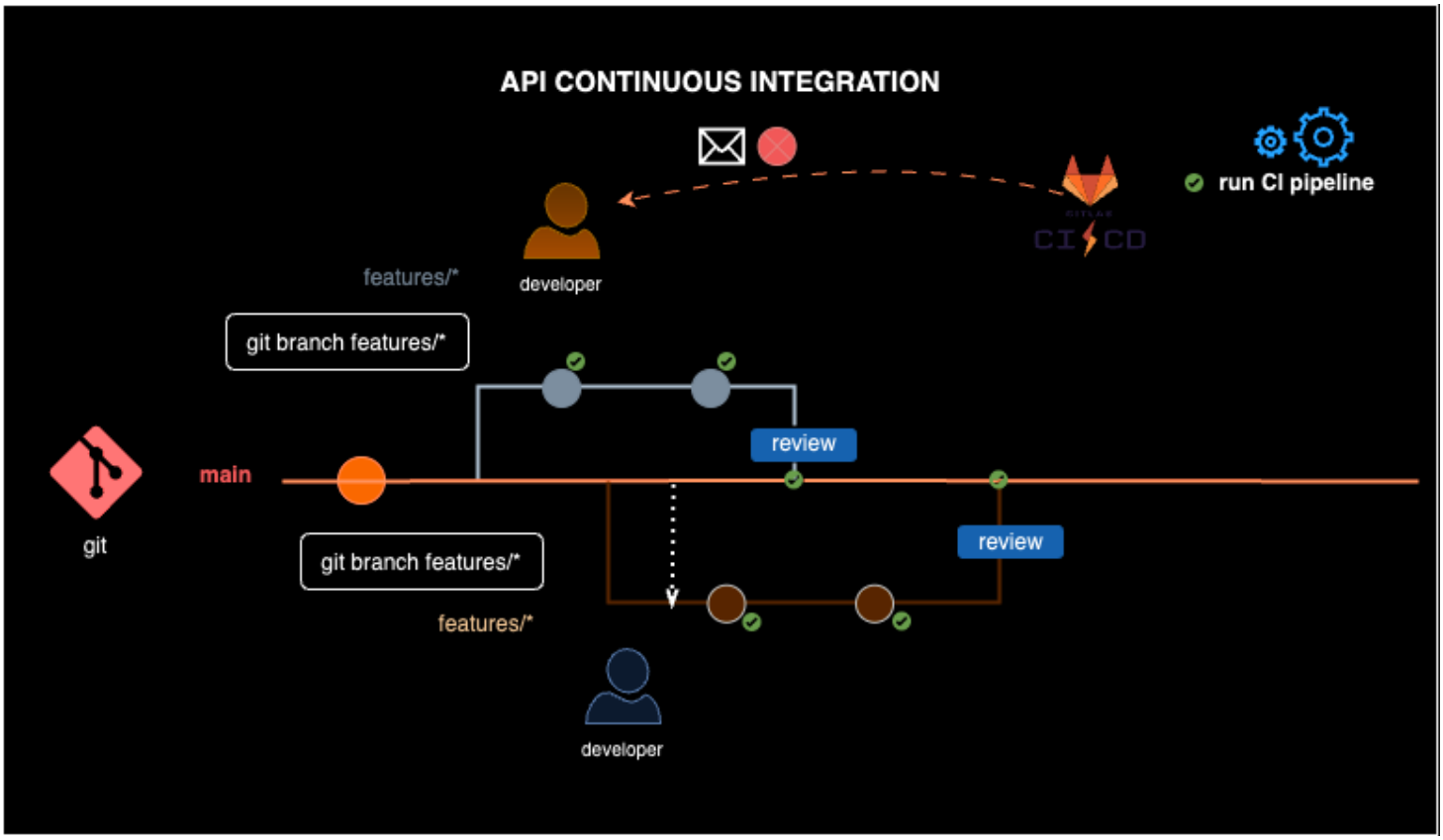
Devamlı ve otomatik olarak bu adımların çalıştırılmasıdır. Bu aynı zamanda DevOps kültürünün pratiklerinden biridir. En çok kullanılan CI build serverlardan **Jenkins**, **GitLab**, **CI-CD** ve **Github actions** gibi örnekler verilebilir..

Şimdiye kadar anlattıklarımı uzun bir çizgi üzerinde gösteriyorum tekrardan. Yukarıda da hangi alanın continuous integration olduğunu belirtiyorum. Evet şimdiye kadar ki adımlarla kaynak koduna yapılan her değişikliği kontrol ettik, test ettik ve analiz ettik ama fark ettiyseniz henüz bir ortama kurmadık ve çalıştırmadık. İşte bunun yapılacağı sürece giriyoruz.



Bu süreç başladığında o an hazır olan kod kullanılarak geniş bir şekilde **entegrasyon testleri** yapılır. Entegrasyon testi başarılı olduğunda **release** süreci başlar. Bu sefer geçici değil de gerçek bir paket hazırlanır. Yani artık **production** ortamı için ve son kullanıcıya açmak için bir paket hazırlanır bu pakette yine her **Artifact repository** sistemine gönderilir. Bir sonraki deploy adımında ise, paketlenmiş bu son sürüm şirket içinde belirlenen ortamlara bir süreç dahilinde kurulur. İşte bu en baştan buraya kadar olan sürece verilen isim
`Continuous Delivery ve Contiuous Deployment{:html}` tır. Devamlı bir paket teslim etme ve o paketin otomatik olarak ortamlara kurulması manasındadır.

2.6 Release



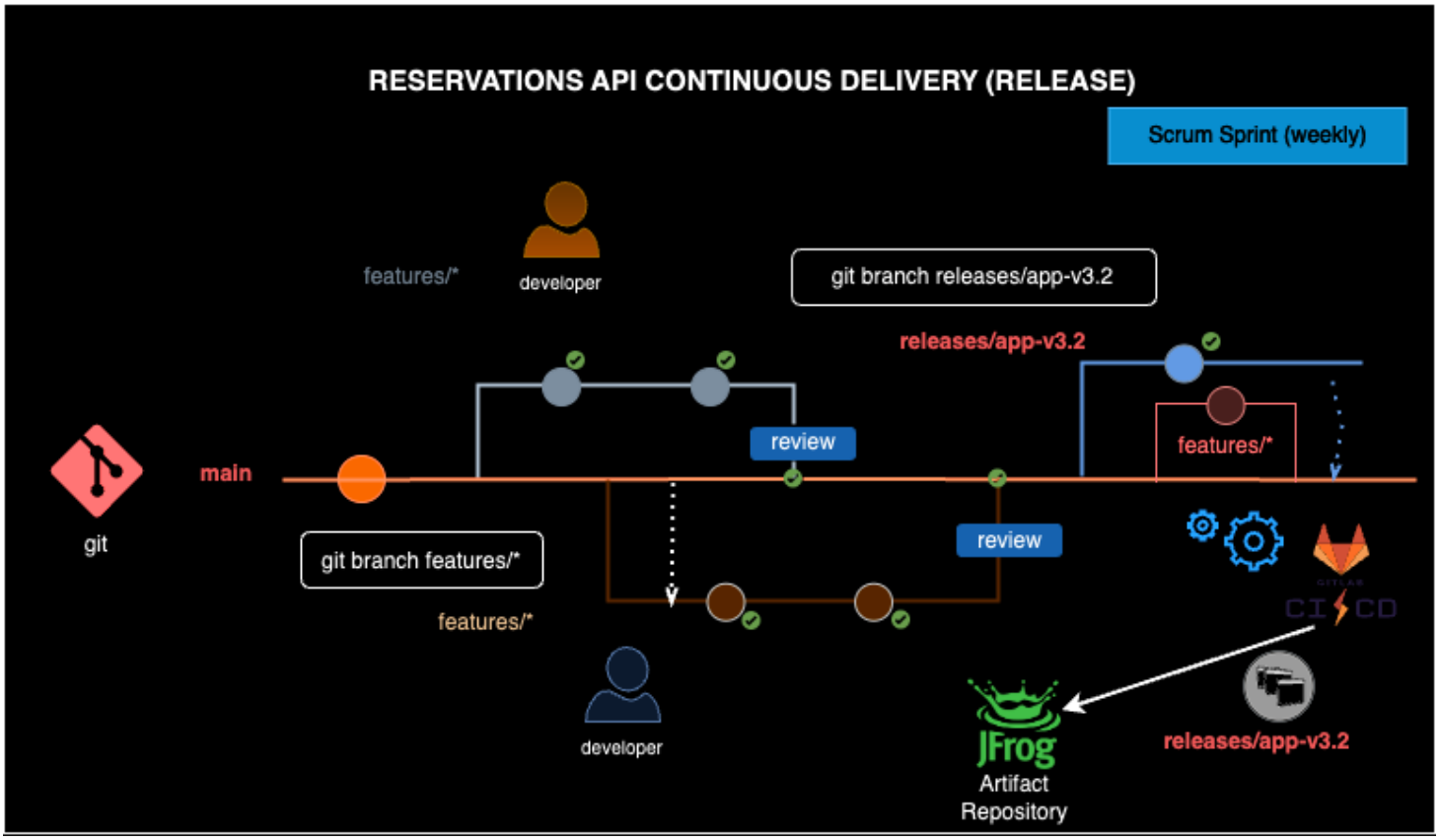
Git içinde olan bir projenin her zaman bir ana branch yani bir ana dalı olur. Git sistemi, içinde farklı branchlar yani dallar ile çalışır. Bu ana brancha verilen isim **masterdır**. Bazen **main** olarak da isimlendirilir. Branchlarla çalışmak her şirkette aynı olmayabilir.
Master branchta olan kod her zaman tamamen test edilmiş hatasız çalışan kod olması gerekiyor. Eğer bu branchta bir hata olursa takım bunu hemen çözmesi gerekiyor. Çünkü diğer yapılacak branchlar master branştan yola çıkılarak yapılır. Mesela diyelim takımda yazılımcılardan biri bir değişiklik yapması gerekiyor. Bunu direk master üzerinde yapamaz, bunun için master baz alınarak yeni bir branch yapıyor, bir tür geçici çalışma branchı.

Bunlara **feature** branch deniyor. Bu tür branch isimleri genelde **features** kelimesi ile başlar. Bu yeni feature branch yaptıktan sonra yazılımcımız kendi branchında değişiklikleri yapıyor. Aynı zamanda takımdaki diğer bir yazılımcı da bir **feature branch** yapıyor ve kendi değişikliklerini kodluyor.

Bir süre sonra ilk yazılımcı sorumlu olduğu değişikliği yapıp test ettikten sonra ana brancha geri yazıyor. Daha sonra ikinci yazılımcı da kendi değişikliklerini ana branche yazmak istiyor. **Fakat ana branch artık ilk baz aldığı gibi değil. İlk yazılımcının değişiklikleri de var. Bundan dolayı ikinci yazılımcı önce master'daki değişiklikleri kendi branchına alıyor, tekrar her şeyi test edip daha sonra master'a geri yazıyor.** Tabii ki master'a gönderilen her değişiklik takımdaki başka en az bir kişi tarafından review yani incelenmesi gerekiyor. Gitlab ve github gibi araçlar bunun en kolay şekilde yapılması için birçok özellik sunmaktadır.

Bu anlatılanlar git tarafında olan değişiklikler, şimdi gelelim build serverın burada nasıl bir işlevi olduğuna. **Git e direkt yazılan her değişiklik build server tarafından kontrol edilir**. fotoğrafta gösterdiğim her yerde yapılan değişiklik build server tarafından otomatik kontrol edilmektedir. Eğer yapılan kontrol sırasında bir hata olursa bu değişikliği yapan kişiye ve bazen de tüm takım bireylerine mail veya bilgilendirme gider. Buraya kadar anlattıklarım continuous intigration kontrollerinin çalışmasını sağlayan durumlardır.

Continuous delivery olayı nasıl çalışır.



Takım içerisinde yazılımcılar proje üzerinde gerekli değişiklikleri yapıp devamlı olarak bunları **master branche** göndermektedir. Ama bu değişiklikleri son kullanıcı henüz görmüyor. **İşte projeden sorumlu kişi takım ile beraber son release'ten bu ana kadar yapılan değişiklikleri son kullanıcıya ulaştırmak için bir release takvimi oluşturur.** Bu takvimde kaç zamanda bir release edilmesi bilgisi yer alıyor. **Çoğu şirket bu release işlemini her Scrum Sprint bittiğinde yapar.**

Scrum yazılım geliştirme ve proje yönetimi süreçlerini hızlandırmak ve esnekliği arttırmak amacıyla kullanılan bir çerçeve ve meteorolojidir. Özellikle karmaşık ve değişken projelerde olmak üzere tasarlanmıştır ve çevik agile geliştirme prensiplerine dayanır. Ekibin düzenli aralıklarla işlevsel ürün parçalarını teslim etmesini ve sürekli olarak geri bildirim almasını hedefler. Scrum içinde Sprint uzunlukları genellikle 1 haftadır bazen de 2 hafta olur. Yani takım 1 veya 2 hafta boyunca önceden anlaşılan değişiklikleri yapmak için koşuya başlıyor diyebiliriz.

Takımlarda release ve deploy işlemini sprint sonunda yaparlar, yani o sprint içinde hangi özellikler bittiyse onları release ederler.

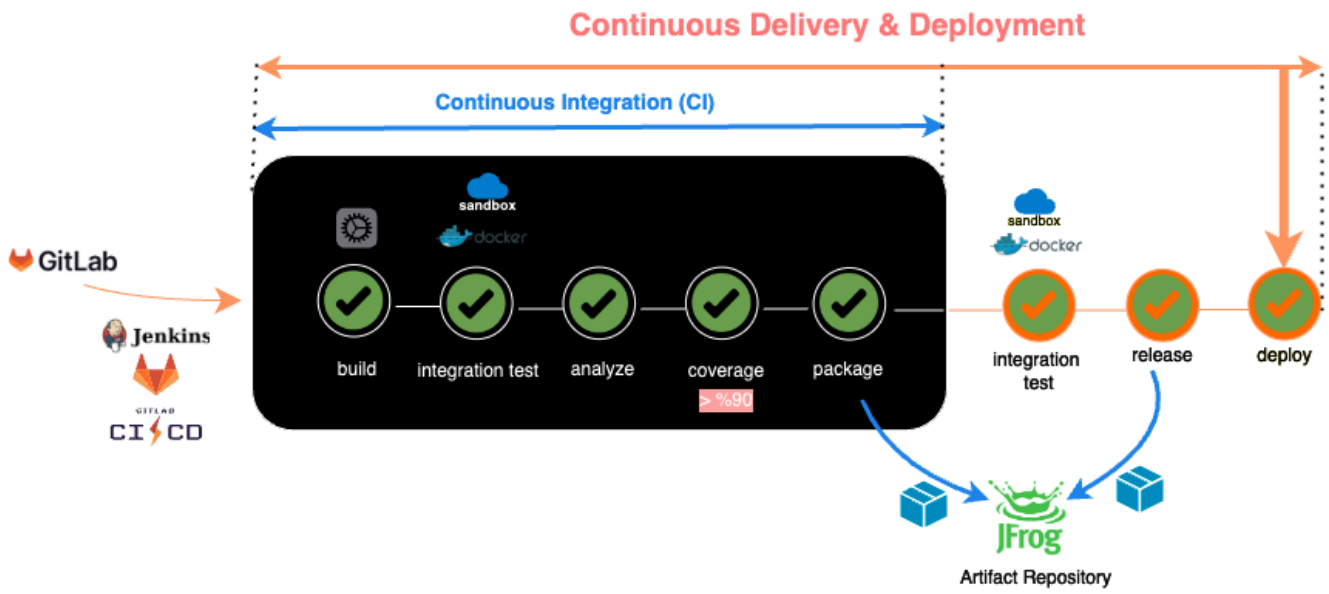
Git üzerinden release işlemi nasıl gerçekleştiriliyor? Kararlaştırılan zamanda **master** üzerinden bu sefer bir **release branch** oluşturuluyor ve artık son kullanıcıya gidecek sürüm bu sürüm olacak. Bu ayrı bir branch olduğu için ekip yine masterda çalışmaya devam edebilir. Eğer test sürecinde **release branch** içinde değişiklikler yapıldıysa bunlar aynı zamanda master'a da gönderilebilir. Daha sonra **CI-CD** süreci bunun içinde çalıştırılıyor.

En kapsamlı şekilde test ediliyor ve tüm adımlar başarılı olduğu takdirde yeni bir **release package** hazırlanıp **Artifact repository** sistemine gönderiliyor. Buradaki paket artık değiştirilemez ve bir versiyon numarasına sahip bir pakettir, yani son kullanıcıya ulaşmak üzere hazırlanan paket. Normalde bir branch master'a gönderildikten sonra silinir, artık kalmasına gerek yoktur. **Fakat bu yeni yaptığımız release branch silinmez.**

Hazırlanan paket son kullanıcıya sunulduğunda muhtemel oluşacak hataları gidermek için yine bu branch kullanılır. **Çünkü master devamlı hareket halinde olduğu ve devamlı değişiklikler aldığı için canlıdaki özelliklerle master'daki özellikler aynı değildir.** Bu yüzden bunları düzeltmek için o anda hangi release branche ait ise o kullanılır.

Bazı şirketler devamlı olarak master'dan çalışmaktadır ve release branch kullanmazlar, diğerleri ise yine çok farklı bir git branching modeli kullanırlar.

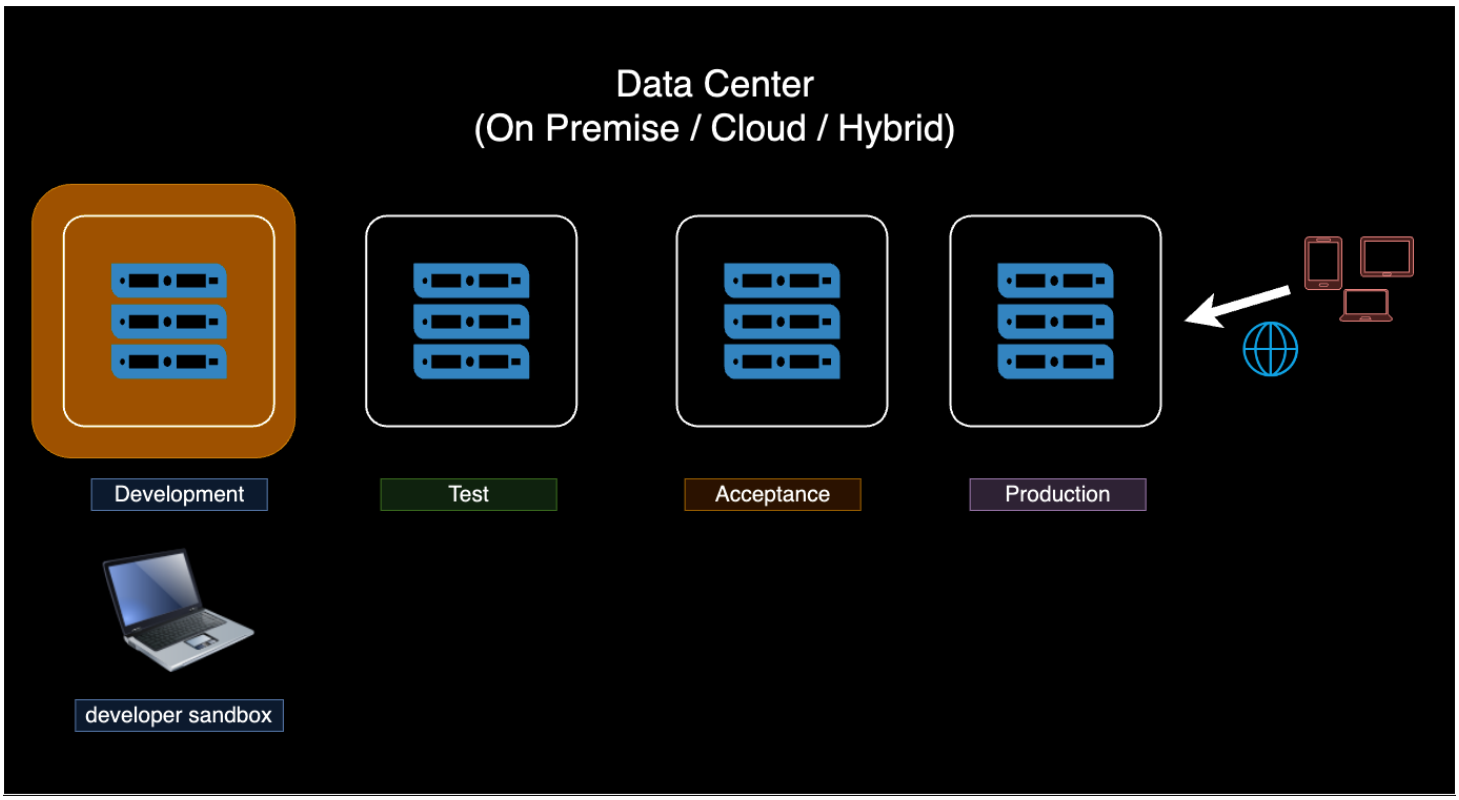
2.7 Deploy



Deploy etmek bir yazılım uygulamasını veya sistemin geliştirme sürecinden üretim veya canlı ortama taşıma işlemidir. Bu işlem genellikle yazılımın kullanıcılar tarafından erişilebilir ve kullanılmaya hazır hale getirilmesini içerir. Yazının başında anlatılan, şirketler uygulamalarını kullanıcılar tarafından erişilebilir kılmak için bu tür data center içinde bulunan makinelere, sunuculara kurmaktadır. Genel kabul olarak şirketlerde 4 tür kullanım ortamı vardır.

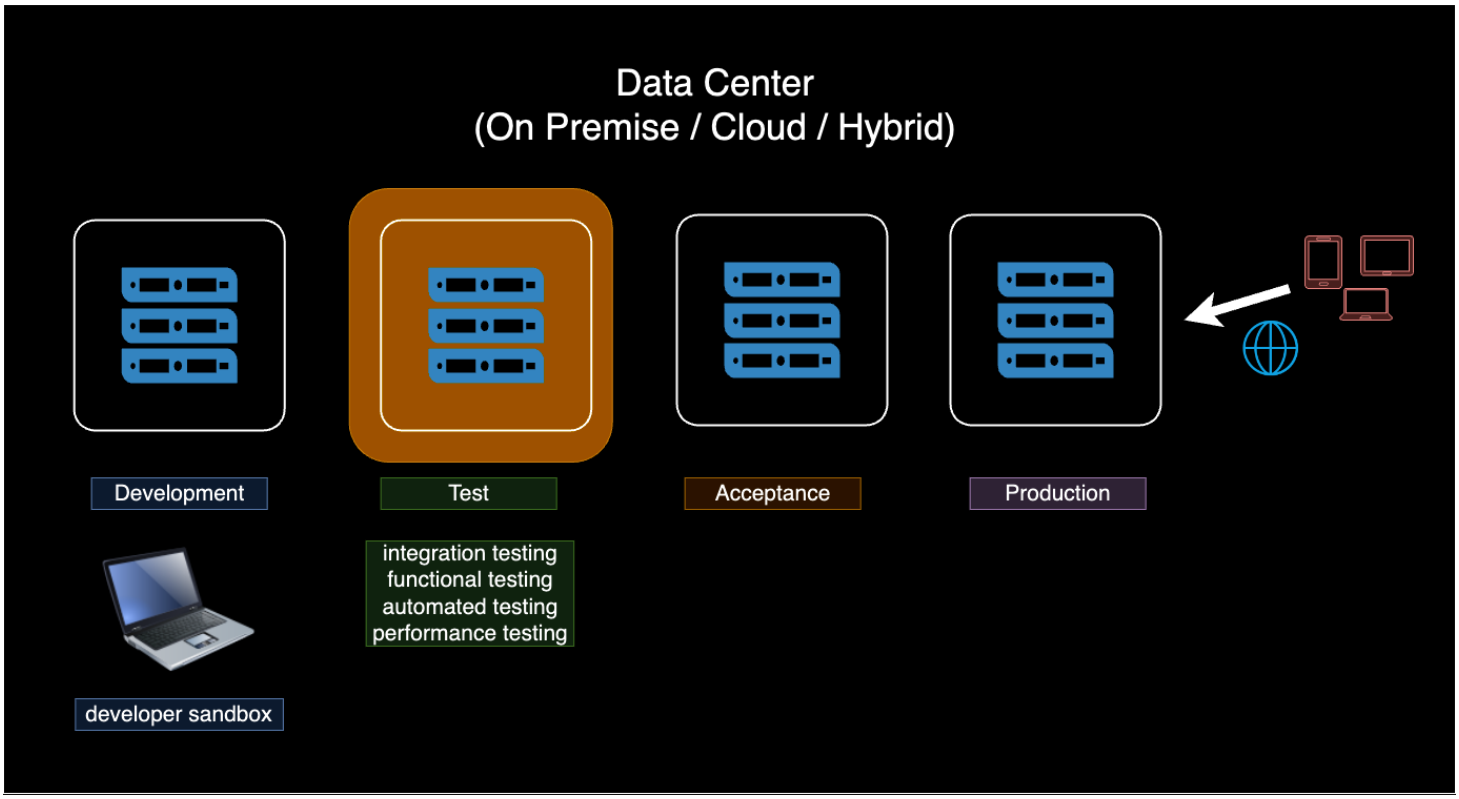
Bunlar **development**, **test**, **acceptance** ve **production** yani canlı. Her bir ortamın görevi farklıdır ve her ortamda bir veya birden fazla bilgisayar bulunur.

2.7.1 Development



Development ortamı yani geliştirme ortamı takım içindeki yazılımcılar için bir nevi oyun alanıdır, yani istedikleri gibi hiçbir şeyin bozulmasından veya çalışmamasından korkmadan istedikleri gibi bu ortamda kodlarını yazılım süreci boyunca test ederler, değişiklikler de yaparlar. Bazı şirketlerde böyle ayrı bir ortam yoktur ve yazılımcının kendi laptopu development ortamı olarak görülür.

2.7.2 Test

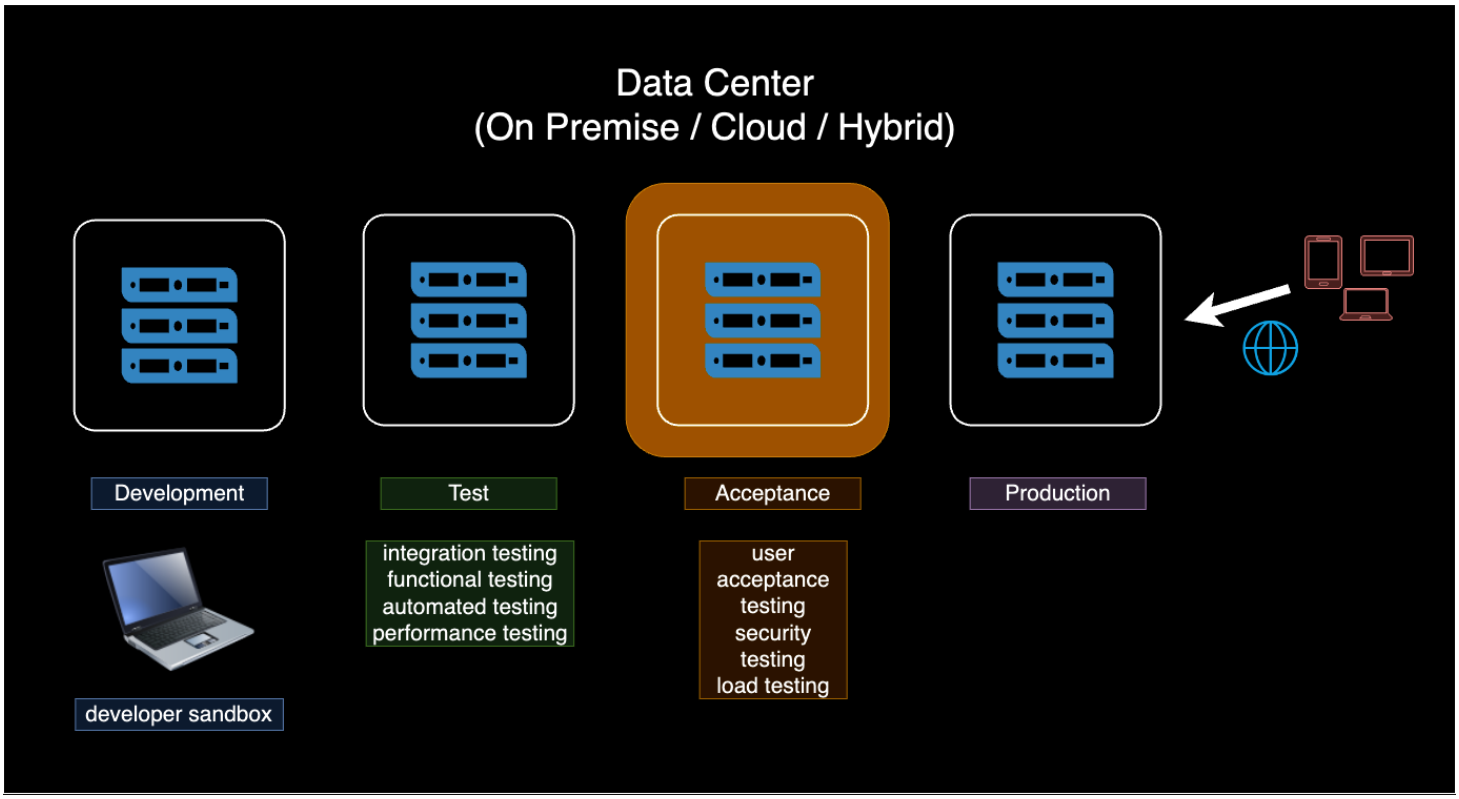


İkinci ortam ise **test environment** başka bir adıyla **entegrasyon** ortamı. Burada artık sizin ve diğer takımların uygulamaları kurulu ve çalışır haldedir. İlk defa bu test ortamında her takımın uygulaması birbiri ile gerçekten etkileşime geçer. Bu yüzden bu ortama entegrasyon ortamı denir. Bu ortamda tüm api'ler birbirine gerçekten bağlanıp test edilir. Daha önceki development ortamında bu etkileşim **mocklar** veya **stublarla** oluyordu, yani diğer uygulama taklit ediliyordu. Örnek olarak reservationsAPI mailServiceApi çağırması gerekiyor. İşte bu test ortamında gerçekten rezervation api uygulamamız mail servisi çağırmaktadır. Bu ortamda genelde yapılan faaliyetler şunlardır:

- **integration testing**,
- **functional testing**. Fonksiyonel olarak yazılım gerçekten beklenildiği gibi çalışıp çalışmadığını otomatik olarak veya takımdaki bir tester tarafından manuel olarak test edilir.
- **Automated testing** ise diğer farklı otomatik yapılan testlerdir.
- **Performance testing** her ne kadar bir sonraki ortamda da yapılıyor olsa da bu ortamda da olabilir. Bu performans testi ile beraber uygulamanın çalışma ve İşleme hızını test edersiniz. Mesela saniyede gerçekten beklenildiği gibi, örneğin 1000 işlem yapabiliyor mu diye.

Bu test ortamında yapılan bunca testin sonucu yetersiz veya hatalı bulunursa tekrardan kod düzeltilip yeniden test edilir.

2.7.3 Acceptance Environment



Bir sonraki ortam ise **acceptance environment**. Eğer bir yazılım bu ortama kadar gelmişse fonksiyonel yönden oldukça hazır durumdadır, artık canlıya çıkmak için en son önemli testler yapılır.

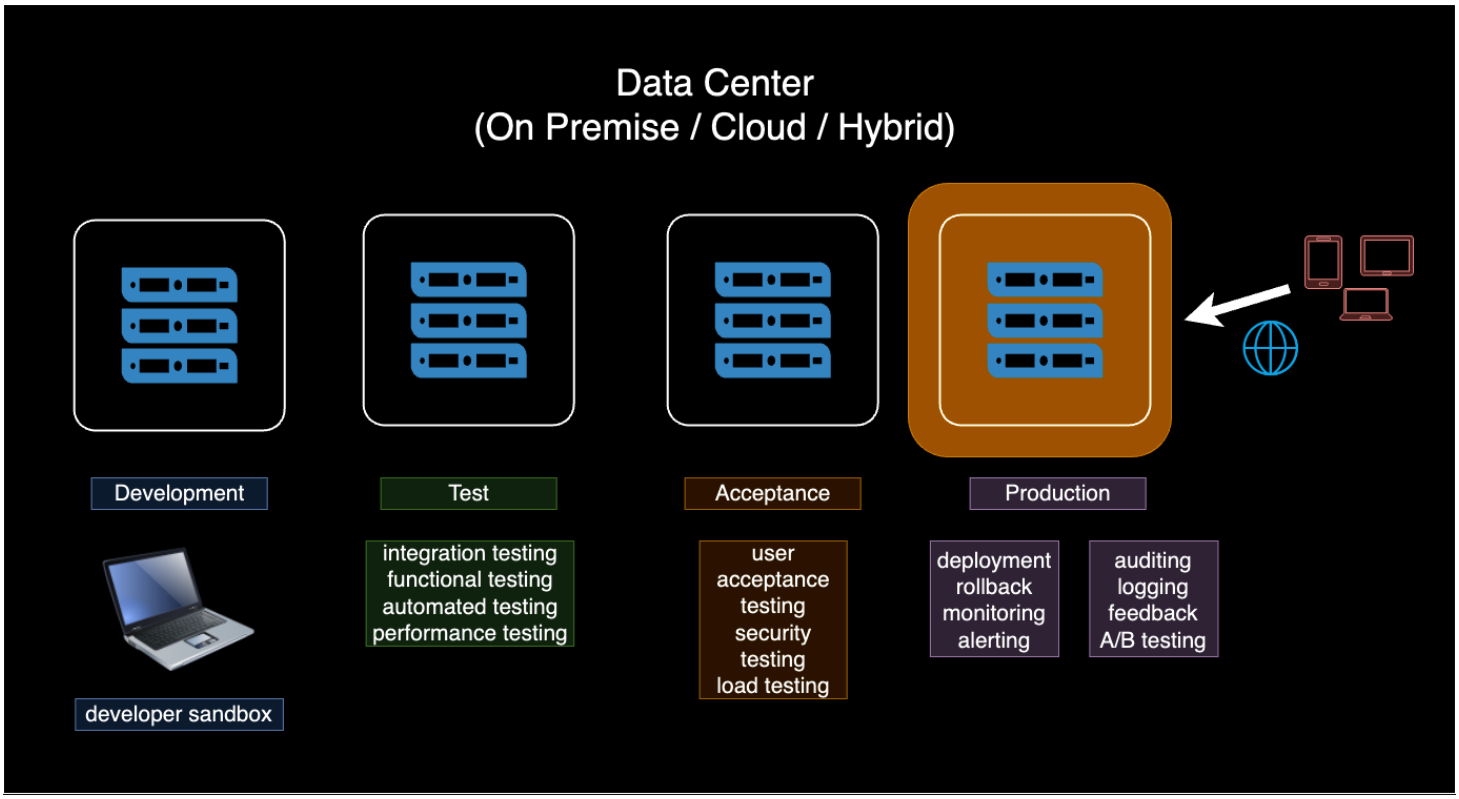
Bu ortam genellikle production ortamı ile aynı özelliklere sahip olur. Bu ortam bir uygulamanın canlıya çıkmadan önce aynı canlı ortam özelliklerine ve konfigürasyonuna sahip bir ortamda test edilmesi için vardır. Aksi takdirde canlıda ortaya çıkan bir hatayı çözmek için aynı öyle bir ortamda test edilip ortaya çıkarılması gerekir.

Bu acceptance ortamında yapılanlar başlıca şunlardır:

- Şirket bünyesinde son kullanıcıları temsil eden kalite kontrol ekibi veya **stakeholders** dediğimiz şirkette doğrudan veya dolaylı olarak ilgisi payı veya hakkı bulunan ve uygulamanın beklenildiği gibi çalışmasının menfaatlerine olacak kişi grup veya işletmedir.
- Bir sonraki test ise **security testing**'dir. Bu da artık yazılımın içeri veya dışarıdan projeye dahil olan bir güvenlik ekibi tarafından her türlü güvenlik açığının kontrol edilmesidir. Bu güvenlik testi bazen de test ortamında da olabilir.
- Başka bir test ise **load testing**'dir. Bunda da uygulamanın performansı ölçülür ama ekstra olarak uygulamanın beklenenden çok daha fazla trafiğe maruz kaldığında nasıl davrandığı görülmek istenir. Mesela yazılan uygulama saniyede 1000 ile 2000 arası çağrıyı işleyebilecek kapasitede yazıldı ise bu test ile uygulama daha yüksek trafiğe mesela saniyede 5000 ya da 10.000 çağrıya maruz bırakılır. Burada görülmek istenen şey uygulamanın çökmemesi ve yanlış işlem

yapmamasıdır. **Uygulamanın çok fazla trafik karşısında çağrıyı cevaplama süresinin çok uzaması veya time-out vermesi normaldir ama çökmemesi gerekiyor.**

2.7.3 Production



En son ortam ise canlı ortam olan **production** ortamıdır. Artık uygulama buraya kurulur ve son kullanıcıların kullanımına sunulur.Sonuçta bir uygulamanın canlıya çıkması yeterli değildir. Ondan sonra nasıl bir **Operations** desteği olduğu önemlidir. Her beklenmedik hata hemen çözülmesi gerekiyor. Aksi takdirde şirketin itibarı zedelenebilir ve kullanıcılarını kaybedebilir.
Rollback eğer uygulamanın yeni sürümü ortama kurulduktan sonra beklenmedik bir hata olursa, hızlıca bir önceki sürümüne dönebilmesi için gerekli hazırlığın yapılması gerekiyor. Bu olay gerçekten çok yaşanmaktadır. Mesela biz 3.2 sürümünü kurduk ama beklenmedik hatalar oluşuyor hemen 3.1 yani önceki versiyonuna dönmesini sağlıyoruz.

Monitörün ve **Alerting**. Bunlar bir sistem veya uygulamanın performansını ve durumunu izleme ve sorunları tespit etme süreçlerini ifade eder.

Monitoring, bir sistem veya uygulamanın belirli durumları veya performans ölçütlerini düzenli aralıklarla izleme sürecidir. Bu metrikler cpu kullanımı, bellek kullanımı, ağ trafiği, kullanıcı etkileşimi ve daha birçok şeyi içerebilir.

Alerting ise bir sistem veya uygulamada belirli bir durum veya metrik belirli bir eşiği açtığında otomatik olarak uyarı verme sürecidir. Bu uyarılar genellikle sorunların Hemen fark edilmesine ve müdahaleye olanak tanır. Her takım kendi uygulamalarını nasıl çalıştığını canlı olarak takip edebildiği

birçok grafikler, şemalar, ekranlardan oluşan dashboard'lar hazırlarlar. Mesela bir makinede sabit disk doluluğu %90'a geldiğinde hemen mail, sms veya farklı şekilde uyarılar gönderilir. Aynı şekilde uygulama çalışma sırasında beklenmedik durumlarda olabilir. Mesela yazdığınız uygulamanın çağrılara cevap verme süresi normalde 5 mili saniye ise ve bu sayı birden yüzlere çıktığında hemen yine bir alarm verilir ve takımdakiler haberdar edilir.

Auditing bir bilgisayar sistemine ait kaynakların kullanımını, değişikliklerini ve erişimini izleme ve kaydetme sürecini ifade eder. Bu süreç bir sistemdeki olayları takip etmek, güvenlik politikalarını uygulamak ve gerektiğinde izleme ve raporlama yapabilmek amacıyla gerçekleştirilir. Denetim genellikle bir güvenlik ve uyumluluk önlemidir. Mesela veri tabanında veya makinenin içinde bir değişiklik yapılıyorsa bunun kim tarafından ve hangi tarihte yapıldığı kaydedilir. Eğer yapılan değişiklik güvenlik kurallarına uygun değilse hemen yine gerekli birimlere alarm mesajları gönderilir.

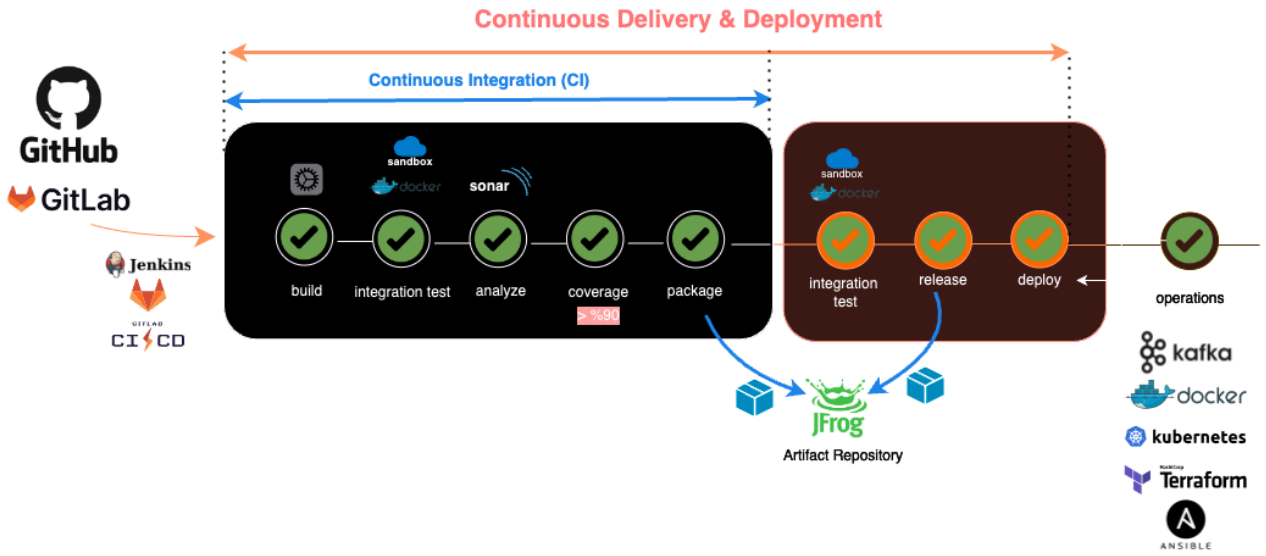
Logging yazılan uygulamamız çalışma esnasında devamlı olarak loglama yapar. Şirketler bu logları her an analiz ederek beklenmedik mesajların veya hataların olup olmadıklarını kontrol ederler.

Feedback bu işlem kullanıcılardan gelen geri bildirimlerin düzenli ve doğru bir biçimde ait oldukları birimlere yönlendirilmesini içerir.

A/B testing bu iki veya daha çok sürümün yan yana çalışarak karşılaştırma testlerinin olduğu testlerdendir. Kullanıcının tepkisine ve kullanışına göre mesela hangi özelliğin seçilmesi gerektiği gibi testler yapılır.

Evet, belki şu an aklımızdan geçiyordur bunca şeyin teker teker yapılması çok zaman alır diye. Ama gerçekte öyle değildir. Artık birçok adım otomatikleştirildiğinden dolayı insan eli değmeden her şey olması gerektiği gibi yapılmaktadır. Mesela bazı şirketler her ay yeni bir sürümü kullanıcıya çıkartırlar ama Google Facebook gibi büyük şirketlerde çoğu zaman her değişikliği anında kullanıcıya çıkartırlar. İşte her şirket farklı olabiliyor. Şimdiye kadar anlattığım tüm bu adımlar ne kadar çok otomatikleştirilirse süreçler o kadar hızlı işler.

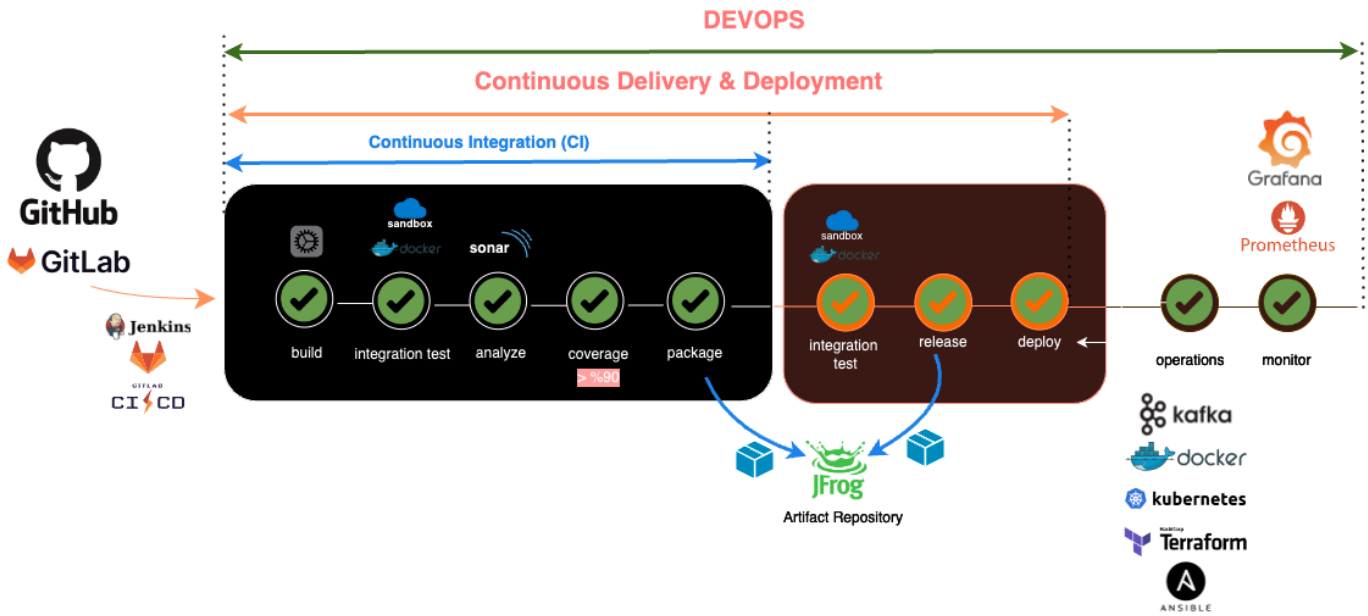
2.8 Operations



Operations adı altında o ortamın her türlü sunucular, ağlar veri tabanları, cpu, hafıza, bellek, işletim sistemi konfigürasyonu, güvenlik sürümleri gibi birçok özelliğin hazırlandığı yerdir.

Bunları otomatik yapabilmek için kullanılan araçlar **terraform**, **ansible**, **kubernetes** ve **docker** gibi araçlardır. Sonuçta sisteme yeni bir makine dahil edilmesi durumunda her şey elle yapılmıyor artık. Terraform ve ansible gibi **Infrastructure As A Code** tarzıyla artık tüm özellikler scripler yazılarak hazırlanıyor. Bir script çalıştırılması ile anında sisteme istenildiği özelliklere sahip bir yeni makine ilave edilebiliyor.

2.9 Monitor



Bunu sağlamak için de en çok kullanılan sistemler **prometheuse** ve **Grafana**'dır.

İşte en baştan buraya kadar anlatılanlan tüm bu süreçlerin toplam ismine verilen isim **DevOps**.

DevOps yazılım geliştiricilerle bilgi teknolojileri operasyonları arasındaki işbirliğini güçlendirmeyi ve yazılım uygulamalarının daha hızlı daha güvenilir ve daha etkili bir şeklide teslim edilmesini sağlamayı amaçlayan bir kültür, işbirliği modeli ve bir dizi pratiktir.

Artık çoğu zaman yazılımcılar ve operasyon ekibi farklı takımlarda değillerdir. Aynı takım içinde beraber çalışmaktadırlar. Yazıda bahsedilen araçlar değil bir çok farklı araçta mevcuttur.