TÜBİTAK BİLGEM YTE | YAZILIM TEKNOLOJİLERİ ARAŞTIRMA ENSTİTÜSÜ

CI/CD SÜREÇ YÖNETİMİ

ŞUBAT 2023 / SAYI: 02

ARAŞTIRMA SERİSİ

Simge ve Kısaltmalar

Kısaltmalar	Açıklama
CI	Continuous Integration / Sürekli Entegrasyon
CD	Continuous Delivery / Sürekli Teslimat
BT	Bilgi Teknolojileri
TÜBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu
BİLGEM	Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi
YTE	Yazılım Teknolojileri Araştırma Enstitüsü

Yazar: Ahmet Ceyhan
Uzman Araştırmacı, Yazılım Geliştirme Bölümü
TÜBİTAK BİLGEM YTE

İletişim: 0(312) 289 92 22 - yte.bilgi@tubitak.gov.tr

yte.bilgem.tubitak.gov.tr

İçindekiler

Önsöz

TÜBİTAK BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü, 2012 yılından bu yana yazılım teknolojilerinde AR-GE faaliyetleri yürüten bir araştırma kuruluşudur. Araştırma faaliyetlerinde elde ettiği birikimini stratejik, hassas ve kritik projeler yürüterek kamu adına hayata geçirmekte; kurumlarımıza dijital dönüşüm, yazılım geliştirme teknolojileri ve kalite süreçleri konusunda danışmanlık vermektedir.

Araştırma Serisi ile TÜBİTAK BİLGEM YTE kurum içi çalışmaların yaygınlaştırılması ve sektörün erişimine açılması amaçlanmaktadır. Araştırma Serisi'nde yayınlanan çalışmalar TÜBİTAK BİLGEM YTE çalışanlarının projelerde elde ettiği bilgi birikimini paylaşmak adına derlenmiştir. Bu çalışmalar ile ülkemizin yazılım sektörüne katkı sağlanması hedeflenmektedir.

Giriş

Son zamanlarda, yazılım geliştirme sektörü yavaş ama gerçek bir dönüşümden geçmektedir. Yazılımlar giderek her şeyin bir parçası haline gelirken yazılım geliştiriciler, artan bu taleple daha fazla otomasyon yoluyla başa çıkmaya çalışmaktadır. Yeni özelliklerin ve uygulamaların hızlı dağıtımına yönelik yoğun talep nedeniyle Sürekli Entegrasyon (CI) ve Sürekli Teslimat (CD) prensipleri giderek daha fazla ve daha etkin kullanılır hale gelmiştir. Geliştiricilerin uygulamalarının hızlı ve güvenilir bir şekilde devreye alınmasını sağlamak için altyapı mühendisleriyle yakın işbirliği içinde oldukları DevOps yaklaşımları uygulanmaya başlanmış ve bu yaklaşım sayesinde projelerin verimliliği büyük ölçüde artmıştır.

Yazılım geliştirmenin ana hedefi ürünü/hizmeti kullanıcıya ulaştırmaktır. Yazılım geliştirme bir süreçtir, yönetilmesi gerekir ve maliyetleri vardır. Yazılım şirketlerinin hedefleri ise ürünün işlevselliği dışında yazılımı en kısa zamanda ve en ekonomik şekilde müşterilerine sunmaktır.

Standart çevik yaklaşım pratikleri her sprint sonu teslimatında yeni özelliklerin sisteme girişini temsil eder. Bu pratikler yeni geliştirilmiş özellikler veya teslimatı etkileyen hataları veya bozuklukları içerebilir. Bu çalışmanın amacı CI/CD pipeline yaklaşımı ile teslimat zaman çizelgesini, test yükü adımlarını ve kıyaslama görevlerini iyileştirerek teslimat sorunlarının üstesinden gelmektir. Birden fazla test adımını entegre ederek sistem kesintisini azaltır ve tüm sürece kararlılık ve teslim edilebilirlik ekler. Kullanılacak yöntem tam otomatize edilmiş şekilde test edilmiş bir sürece sahip olmak anlamına gelir, standardizasyon sağlar ve ayrıca belirsizliği ve tahminde bulunmayı azaltır, kaliteyi garanti eder ve üretkenliği artırır. Bu yaklaşım, otomatikleştirilmiş işlem hatları aracılığıyla çevik tabanlı CI ve CD projelerini oluşturmak, yönetmek, özelleştirmek ve otomatikleştirmek için etkili ve verimli bir yol sağlar. Bu çalışmada anlatılanlar, standart CI/CD süreçleri için bir başlangıç noktası görevi görür, uygulamaların tekrar kullanımı için versiyonlama ve ArgoCD kullanarak bir Kubernetes kümesinde yüksek düzeyde kullanılabilir çıktıları uygular.

1. Temel Kavramlar

1.1. Sürekli Entegrasyon (Continuous Integration, CI)

CI, geliştiricilerin kodun uygulanabilirliği hakkında hızlı bir şekilde geri bildirim almak için, yazdığı kodu sürekli olarak paylaşılan bir kod deposuna entegre ettiği uygulamadır. CI, otomatikleştirilmiş derlemeleri ve testleri destekler, böylece ekipler tek bir proje üzerinde hızla işbirliği yapabilir. Ayrıca CI, yazılım şirketlerinin daha sık ve daha kısa bir sürüm döngüsüne sahip olmalarını sağlar.

Bu strateji, geliştirilen uygulamanın, canlı ortamda hızlı ve güvenilir bir şekilde yayınlanmasını kolaylaştırır. CI, geliştirme ekiplerini kısa yinelemelerde yazılım oluşturmaya ve işlevsel kodlarını mümkün olan en kısa sürede kök kodla birleştirmeye teşvik eder.

1.2. Sürekli Teslim (Continuous Delivery, CD)

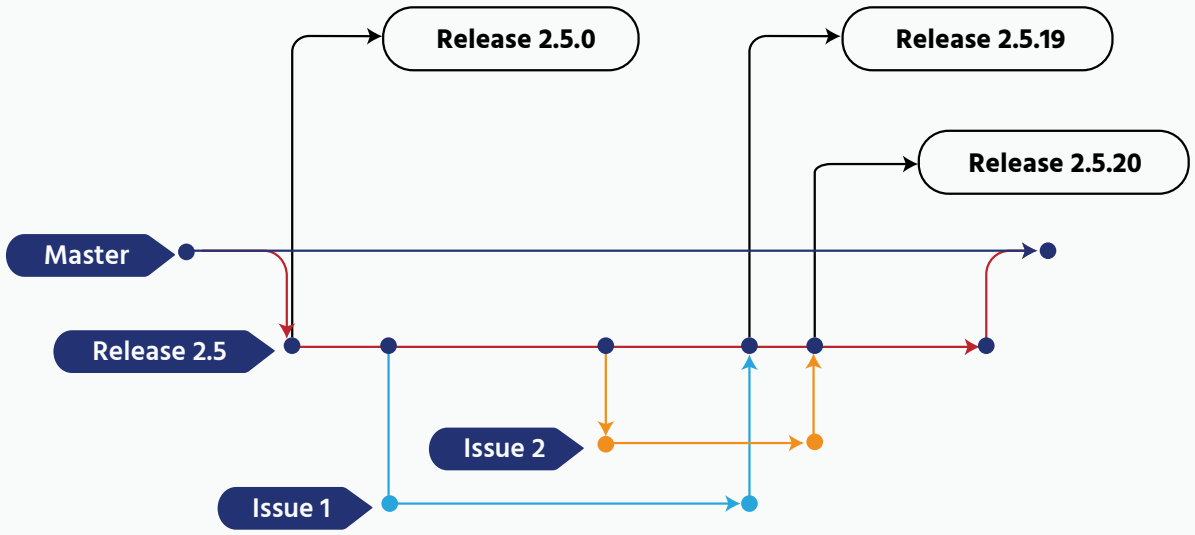
CD, ekiplerin kısa döngüler halinde yazılım tasarladığı, oluşturduğu, test ettiği ve yayınladığı bir yazılım mühendisliği uygulamasıdır. Döngünün hem hızlı hem de güvenilir olmasını sağlamak için her aşamada otomasyona güvenir.

Temel olarak CD, bir uygulama geliştiricisi tarafından yapılan değişikliklerin kod deposuna veya kapsayıcı kayıt defterine teslim edildiği CI sürecininin devamı olan bir süreçtir. Böylece, tüm değişiklikler operasyon ekibi tarafından canlı bir üretim ortamında devreye alınabilir. CD, bunu yaparak DevOps ve yazılım ekipleri arasındaki sınırlı olan iletişim sorununu çözer. Sonuç olarak, yinelenen bir şekilde yeni kodun mümkün olduğunca az operasyonel bağımlılıkla, otomatik olarak test ve canlı ortamlara kurulmasını ve yayınlanmasını garanti eder.

1.3. Sürüm Kontrolü ve Versiyonlama

Kaynak denetimi olarak da bilinen sürüm denetimi, bir yazılımın kodundaki değişiklikleri izleme ve yönetme yöntemidir. Sürüm kontrol sistemleri, geliştiricilerin zaman içinde kaynak kodundaki değişiklikleri yönettiği yazılım araçlarıdır. CI/CD yaklaşımlarıyla en çok kullanılan ve popüler olan sürüm kontrol sistemi Git'tir. Bir veri kümesinin düzenli bir şekilde, genellikle bilgisayar deposunda tutulduğu ve muhafaza edildiği merkezi bir yerdir.

Semantik sürüm oluşturma, yaygın olarak benimsenen bir sürüm şemasıdır; üç parçalı bir sürüm numarası (Major. Minor. Patch), isteğe bağlı bir yayın öncesi etiketi ve isteğe bağlı bir derleme meta etiketi kullanır. Yazılım geliştirmede versiyonlama çok önemlidir. Kök kodu "Master" veya "Main" isimlendirmesi ile isimlendirilebilir. Sprint sonlarında çıkılan her yeni release için Minor numarası arttırılır ve Patch numarası sıfırlanır. Ara sürümlerde ise Patch numarası arttırılır.



Şekil 1. Kaynak Kodun Dallanması ve Ara Sürüm Yönetimi

Ara sürümler, daha çok anlık düzeltmeler için çıkılır. Canlıdaki uygulamada tespit edilen hataların hızlı bir şekilde giderilmesi gerekmektedir. Hatanın sebebi bulunup çözüldükten sonra build edilip otomasyon testleri çalıştırılır ve Patch numarası bir artırılarak etiketlenip imajı oluşturulur. Demo ortamında da kontrolü yapıldıktan sonra canlı ortama alınır.

Ana sürümler, büyük işlevlerin eklenmesi sonunda çıkılır. Uygulama otomasyon testlerinden geçtikten sonra imajı oluşturulur. Yeni release etiketi ile etiketlenir. Önce demo ortamına kurulur. Uygulamanın genel kontrolleri ve yeni eklenen işlevlerin kontrolleri yapılır. Eğer bir hata tespit edilirse düzeltme yapılarak demo ortamına tekrar kurulur. Tüm testler sonucunda hata tespit edilmezse canlı ortama alınır.

2. Ardışık CI/CD Süreçleri ve Avantajları

CI/CD ardışık süreci, planlama stratejisi, geliştirme ve dağıtım anlamına gelir. Bir kuruluş bir CI/CD ardışık süreci benimsemeye çalıştığında, artık bunu bağımsız olarak üstlenemez. İlk olarak, CD'yi benimsemek için CI alıştırması yapmalıdır. CI'den CD'ye aktarılırken, ardışık süreç manuel yürütmeyi azaltır ve tüm yöntem otomatik hale gelir. CD'yi benimserken tüm aşamalar otomasyon yoluyla gerçekleştirilir.

CI/CD ardışık sürecindeki tüm adımlar sırayla ve tümüyle gerçekleşecek şekilde tasarlanmıştır. Önce CI sonrasında CD işletilir. Birçok avantaj sunar.

2.1. Verimli Yazılım Geliştirme

Daha küçük yinelemeler (adımlar), daha kolay ve daha verimli test yapılmasını sağlar. Her yeni adımdaki sınırlı kod kapsamı ve bunu test etmek, hataları bulmayı ve düzeltmeyi kolaylaştırır. Özellikler, kullanışlılık ve kullanıcı kabulü açısından daha kolay değerlendirilir ve daha kullanışlı özellikler, geliştirmeler boşa harcanmadan kolayca ayarlanır.

2.2. Rekabetçi Yazılım Ürünleri

Geleneksel yazılım geliştirme yaklaşımları aylar veya yıllar alabilir ve resmileştirilmiş spesifikasyonlar ve gereksinimler, değişen kullanıcı ihtiyaç ve beklentilerine pek uygun değildir. CI/CD geliştirme, geliştiricilerin değişiklikleri sonraki yinelemelerde uygulamalarını sağlayan yeni ve değişen gereksinimlere kolayca uyum sağlar. CI/CD ile yazılımlar ürünler pazara daha hızlı ve daha başarılı bir şekilde ulaşabilir.

2.3. Başarısız Olma Özgürlüğü

CI/CD'nin hızlı döngüselliği, geliştiricilerin geleneksel yazılım geliştirme yöntemlerinden çok daha az riskle yenilikçi kodlama stilleri ve algoritmalarla denemeler yapmasına olanak tanır. Bir geliştirme işe yaramazsa, muhtemelen hiçbir zaman kullanıma alınmayacaktır ve bir sonraki hızlı yinelemede geri alınabilir. Rekabetçi yenilik potansiyeli, kuruluşların CI/CD kullanması için güçlü bir itici güçtür.

2.4. Daha İyi Yazılım Bakımı

Geleneksel yazılım geliştirmede hataların düzeltilmesi haftalar veya aylar alabilir, ancak bir CI/CD ardışık sürecinin sürekli akışı, hataların daha hızlı ve daha güvenli bir şekilde ele alınmasını ve düzeltilmesini kolaylaştırır. Ürün her zaman daha kararlı ve güvenilirdir.

2.5. Daha İyi Operasyon Desteği

Düzenli yazılım sürümleri, operasyon personelini yazılımın gereksinimleri ve izleme gereksinimleriyle uyumlu tutar. Yöneticiler, daha az dağıtım hatası ve gereksiz sorun giderme ile yazılım güncellemelerini daha iyi dağıtabilir ve geri alma işlemlerini gerçekleştirebilir. Benzer şekilde, BT otomasyon teknolojileri, kurulum veya yapılandırma hatalarını azaltırken dağıtımların hızlandırılmasına yardımcı olabilir.

3. CI/CD Araçları

3.1. Jenkins

Jenkins, pipeline oluşturmada en öncelikli araçtır denilebilir. Java ile yazılmıştır. Çoğu SCM aracını destekler. Oldukça fazla işlevi bulunan ve tüm CI/CD süreçlerini otomatize edebilecek güce sahip olan Jenkins'e bu gücü linux komut satırı betiklerini çalıştırabilmesi verir. Temel düzeyde, yapılmak istenen bir görevi bir hat boyunca çalıştıracak bir web arayüzü vardır. İleri düzey kullanımlarda Groovy dilinde yazılmış pipeline adımları kullanılır. Jenkins yazılım geliştirmesinden gelen kod değişikliklerini takip edebilir ve otomatik olarak pipeline'ı işletir. Zamanlanmış görevler ile pipeline'ları çalıştırabilir. Pipeline'da unit test, integration test gibi testleri koşturup ve docker imajı oluşturarak imajın bir imaj deposuna gönderilmesi vb. adımları içerir. Her kod değişikliği için çalıştırılan pipeline'lar CI sürecinin sağlıklı şekilde ilerleyip ilerlemediğinin kontrolünü sağlar. CD süreci için ArgoCD veya FluxCD tercih edilebileceği gibi önerilen pratikler arasında olmasa bile bash betikleri ile CD süreçlerini de devralıp GitOps sürecine dahil olabilir.

Jenkins; geliştirme, operasyon ve güvenlik araçlarını ve çalışmalarını tek bir uygulamada toplayan eksiksiz bir DevOps platformudur. Ekiplerin yazılım teslimini haftalardan dakikalara hızlandırmasına yardımcı olurken, geliştirme maliyetlerini ve güvenlik risklerini azaltır.

3.2. Kubernetes

Kubernetes; temelde uygulamaların dağıtımını, ölçeklenmesini ve yönetimini otomatikleştirmek için yazılmış açık kaynaklı bir konteyner orkestrasyon aracıdır. Uygulamaların günden güne konteynerleştirilmesi ile Kubernetesin dokunduğu alan giderek artmıştır ve öyle ki sunucular üzerinde toptan bir veri merkezi işlevselliğine kadar gidebilmektedir.

Kubernetes birçok nesneden oluşur. Bu nesneler tanım olarak YAML formatındadır ve GitOps pratiklerine göre GIT deposunda yaml dosyaları olarak saklanır. Bu dosyalar GitOps sürecinin sonunda test veya canlı Kubernetes ortamlarına kurulur. Kubernetesin bu kadar kabiliyetli olmasının nedenleri, kendi içerisinde sanal ağ yapılandırması ve ağ politikalarıyla konteynerlerde çalışan uygulamalara dışarıdan bir hizmete bağımlı kalmada ağ ve güvenlik hizmeti verebilir durumda olmasıdır. Ek olarak kurulabilir veya dışarıdan entegre edilebilir depolama sistemi yönetimi yetenekleriyle, uygulamaların kalıcı disk ihtiyaçlarını da çözebilir duruma gelmiştir. Bilişim ekosistemi Kubernetes kullanımını artırdıkça daha fazla kullanım durumu ortaya çıkmaktadır.

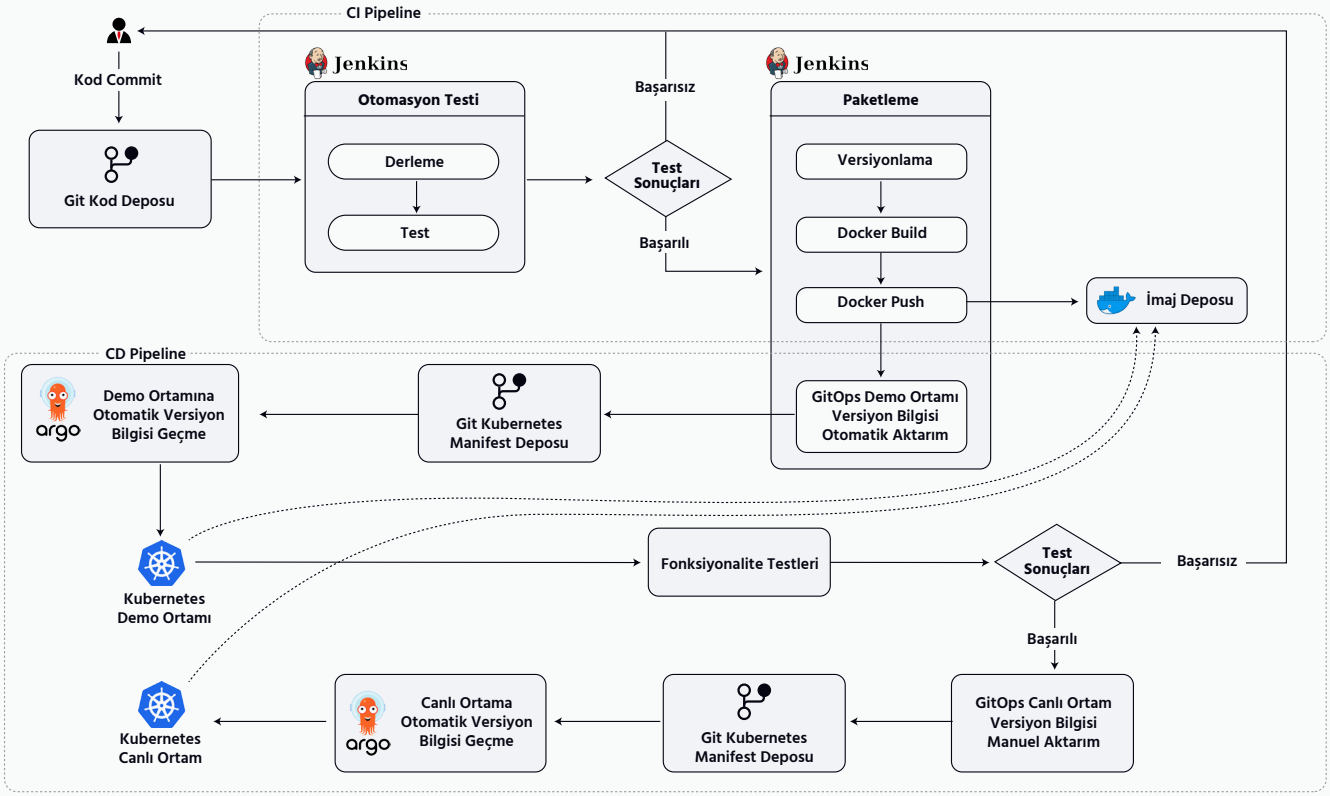
3.3. ArgoCD

ArgoCD, Kubernetes için açık kaynaklı bir GitOps sürekli teslim aracıdır. ArgoCD git deposunda saklanan kubernetes yaml yapılandırma dosyalarını Kubernetes ortamlarına otomatik olarak dağıtır. Git üzerinden sürüm kontrolünün denetlenebilir olmasını sağlar. Hem Git'ten gelen kaynağın hem de hizmete alınan uygulamanın durumunu izleyerek bir Kubernetes denetleyicisi görevi görür. İkisi arasındaki farkları arar ve dağıtılan yazılım uygulamasını manuel veya otomatik olarak güncelleme imkanı sağlar. Bunun yanı sıra, web arayüzü üzerinden farklılıkların mevcut durumunun görselleştirilmiş bir raporunu da sunar. ArgoCD ayrıca Keycloak kimlik yönetimi aracını da destekler ve rol tabanlı erişim kontrolüne sahiptir. Geliştirici ekiplerine, kendi projelerine erişim ve kurulum için yetkilendirme sağlar. Ayrıca Helm desteği de bulunmaktadır.

ArgoCD, bir Git sunucu ve Jenkins ile birlikte Kubernetes üzerinde koşan iş yükleri için tüm yaşam döngüsünü yönetebilir.

Uzmanlar, gelecekteki standart yazılım geliştirme süreçlerinin, yazılım geliştirmeden başlayarak, CI/CD süreçlerini ve gözlemlenebilirliği içeren GitOps sürecini içermesi gerektiğini önermektedir. Özellikle süreklilik isteyen firmalar için tüm yazılım yaşam döngüsünü kapsayabilecek çözümlerin olması çok önemlidir.

4. Yöntem



Şekil 2. CI/CD Yaşam Döngüsü

CI/CD yaklaşımı bir döngüdür. Şekil 2'de gösterildiği gibi bu döngünün her yeni özellik veya hata gidermede gerçekleşecek şekilde tekrar etmesi beklenir. CI ve CD ardışık süreci ile yazılım geliştirme sürecinde tekrar tekrar yapılması gereken işlemler tam otomatik hale getirilir. Yazılımın yapılacak iş için ayrılmış kodunda yapılan her değişiklik ile birlikte CI pipeline'ının başlangıcı olan otomasyon testleri çalışır. Başarısız olursa kod gözden geçirilir ve düzeltme yapılır ve tekrar otomasyon testleri otomatik çalışır. İsteğe göre manuel ya da otomatik işletilen bu adımda versiyonlama birinci önceliktir. Çünkü verilen versiyon numarası ile etiketlenen uygulama imajının kurulumu yapılacaktır. Daha sonra uygulama imajı oluşturulur ve imaj deposuna gönderilir. Ayrıca bu pipeline'da son olarak CD sürecini de içeren GitOps adımı başlar. Oluşturulan imajın semantik sürüm numarası, demo ortamının Git'te tutulan Kubernetes manifestolarında güncellenir. Bu Git deposunu dinleyen ArgoCD otomatik olarak Kubernetes ortamında nesne tanımlarını günceller ve Kubernetes buna göre imaj deposundan imajı çeker ve yeni uygulamayı çalıştırır. Demo ortamında fonksiyonalite testleri yaptıktan sonra başarısız sonuç alınırsa tekrardan en baştaki adıma dönülerek kodda düzeltme yapılır. Canlı ortama geçilmeden önce bir demo ortamında kontrollerin yapılması çok önemlidir. En ufak hata ile tüm veri bozulabilir ya da geri dönüşü olmayan hatalı işlemler yapılabilir. Başarılı sonuçlanan fonksiyonalite testleri sonrası demo ortamı için uygulanan CD nin son adımındaki gibi Git'te tutulan Kubernetes Canlı ortam manifestosu da güncellenir. Uygulamanın güncel versiyonu Canlı ortama kurulmuş olur. Canlı ortamdaki eski uygulama kapatılır. Olası hata durumlarında eski uygulamaların versiyon numarası git kubernetes manifest deposuna geçilmek istenen eski versiyon bilgisi girilir ve eski versiyonlara hızlı bir şekilde geçiş yapılabilir.

4.1. Konteynerleştirme (Docker)

Test otomasyonu düzenlemesi, birim(unit), işlevsel(integration) ve performans testi aşamalarını içerir. Sürekli test yaklaşımı, istikrarlı bir kod tabanı, daha hızlı yanıt ve kolay karar verme avantajına sahiptir. Bu otomasyon olmazsa olmazdır. Her zaman tekrar tekrar işletilir. Build oluşturma adımında, diğer adıyla Sürekli Entegrasyonda en sık tercih edilen konteynerleştirme teknolojisi olan Docker kullanır.

Geliştirme sırasında iş akışı karmaşıktır ve çoğu zaman bir deneme-yanılma sürecine dayanır. Geliştirici bir CI ortamı oluşturduğu zaman tüm Docker konteyner oluşturma adımlarını uygulayabilir: tanımlamaları düzenleme (örneğin, bir Docker dosyası), bir docker imajı oluşturma, beklendiği gibi çalışıp çalışmadığını anlama, bir hatanın neden(ler)ini belirleme ve ilk adıma geri dönüp tanımlamaları geliştirme. Docker, önbelleğe alma, özelleştirme, ölçekleme ve güvenlik gibi özellikler sağlar ve platformlar arası çalışır. Docker'ın bir başka avantajı, özel kontroller kullanarak, genel yürütme süresini azaltmak ve iş görevlerinde kesintileri önlemek için aynı anda çalışabilen birkaç Docker konteynerine karar vermenin mümkün olmasıdır.

```
FROM openjdk:17-jdk-alpine
COPY ./target/demo-app-0.0.1-SNAPSHOT.jar demo-app.jar
CMD [“sh”, “-c”, “java -jar demo-app.jar”]
```

Yukarıda bir java uygulaması için örnek Dockerfile bulunmaktadır. Uygulamanın konteyner haline getirilmesi için bu dosya kullanılır. İçindeki her satır bir komutu temsil eder. İlk satırda base imaj olarak uygulamanın çalışacağı java versiyonuna sahip alpine jdk 17 base imajı seçilir. İkinci satırda derlenmiş uygulamanın jar dosyası ana dizine kopyalanır. CMD ise her Dockerfile da sadece bir tane olur. Docker imajının varsayılan çalıştırma komutunu değiştirmek için kullanılır. Bu Dockerfile örneğinde de java uygulaması için çalıştırma komutu gösterilmiştir.

4.2. Otomasyon Testi

Yazılım geliştirme sürecinde yapılan her kod değişimi ile birlikte Jenkins tarafından otomatik build pipeline'ı çalıştırılması çoğunlukla seçilen yöntemdir. Bu pipeline daha çok ünite ve entegrasyon testleri gibi otomasyon testlerini içerir.

```
pipeline {
  agent any
  stages {
     stage('Checkout code') {
        steps {
           checkout scm
        }
     }
     stage('Test') {
        tools {
           jdk "Java 17"
           maven "LOCALMAVEN"
        }
        steps {
           sh 'mvn clean test'
        }
     }
  }
  post {
     always {
        cleanWs()
     }
  }
}
```

Örnekte bir Jenkinsfile dosyası içeriği bulunmaktadır. Java uygulamasının git deposunu dinleyen Jenkins, her commit sonrası örnekteki Jenkinsfile'a göre pipeline'ı çalıştırır. 'Checkout code' adımında uygulamanın kodu Git deposundan çekilir. Test adımında ise ilk olarak uygulamanın ortam bilgileri belirlenir. Java dilinde yazılmış bir maven projesi olduğu için bu pipeline'ın çalıştığı Jenkins sunucusunda daha önceden kurulmuş olan "Java 17" jdk olarak seçilir ve maven için "LOCALMAVEN" verilir. Daha sonra 'mvn clean test' ile uygulama build edilir ve maven testleri çalıştırılır. Pipeline'ın başarılı olup olmadığına bakılmaksızın cleanWs() ile de workspace temizlenir. Bu pipeline ile uygulamamızın build olması test edilir ve otomasyon testleri otomatik çalıştırılır. Sonuca göre yazılan kodda revize edilmesi gereken yer tespit edilebilir. Örnekteki yöntemde jenkins'in yerel uygulamaları kullanılmaktadır. Bir başka yöntem ise docker imajı oluşturmak ve 'mvn clean test' komutunu docker imajı içerisinde çalıştırmaktır.

4.3. Paketleme

Aşağıda package için bir Jenkinsfile örneği verilmiştir. Bu Jenkinsfile'a göre bir package pipeline'ı çalıştırılır. Pipeline başlangıcında bulunan 'agent any' ile pipeline'ın hangi jenkins makinesi üzerinde çalışacağı seçilir.

'JIRA Version Check' adımında Jenkinsfile'ın en üst kısmında yazılan bir fonksiyon olan jiraVersionCheck() fonksiyonu çağrılır. Jira üzerindeki uygulama versiyonları ile kıyaslama yaparak versiyonlamasının uygunluğu kontrol edilir.

'Package' adımında bash scriptleri ile uygulamanın docker imajı oluşturulur ve container bir imaj deposuna gönderilir. Docker imajı alınan ve kurulmak üzere paketlenen uygulamanın kod içeriğini gözlemleyebilmek için 'Create git tag' adımında git deposu versiyon numarası ile işaretlenir. Bu sayede hangi versiyon numarası hangi kod içeriğine sahip gözlem yapılabilir.

Son adımda jenkins pipeline'ının başarılı olduğu durumda bir başka jenkins pipeline'ı olan sonar job'ı çalıştırılır. Özel olarak hazırlanan gitOpsDeploy() ile uygulama versiyon numarası bir 'sed' komutu ile kubernetes nesnelerinin tanımlamalarının saklandığı "https://bitbucket.bilgem.gov.tr/.../kubernetes-manifests.git" adresli git deposuna yazılır. Başarılı veya başarısız sonuçlanan pipeline'ların sonucunda cleanWs() ile de workspace temizlenir.

```
def jiraVersionCheck() {
  def versions = jiraGetProjectVersions idOrKey: 'DEMOPROJECT', site: 'LOCAL'
  def released_version_list = versions.data.findAll { it.released == true &&
it.name.equals("demo-app-v${VERSION}".toString()) }
  if (released_version_list.isEmpty()){
      currentBuild.result = 'ABORTED'
      error("demo-app-v${VERSION} released statusunde jira versionu
            bulunamadı ... ")
  }
}

def gitOpsDeploy() {
  script {
     dir("k8smanifest"){
        git (
           url: 'https://bitbucket.bilgem.gov.tr/ ... /kubernetes-manifests.git',
           credentialsId: '0017a826-7557-4ed3-9d28-2fb4a00a4e6f',
           branch: 'master'
           )

        sh 'sed -i "/^.*containers:/{n;n;s/image:.*/image: yte.bilgem.gov.
            tr:9092\\/docker-images\\/demo-app:v${VERSION}/;}" demo/deployment/
            demo-app-deployment.yaml'

        withCredentials([gitUsernamePassword(credentialsId:
           '0017a826-7557-4ed3-9d28-2fb4a00a4e6f', gitToolName: 'Default')]) {
           sh 'git add demo/deployment/demo-app-deployment.yaml'
           sh 'git commit -m "demo app versiyonu v${VERSION} olarak
               guncellendi"'
           sh 'git push --set-upstream origin master'
        }
     }
```

```
  }
}

pipeline {
 agent any
 stages {
    stage('JIRA Version Check') {
       steps {
          jiraVersionCheck()
       }
    }
    stage('Checkout code') {
       steps {
          checkout scm
       }
    }
    stage('Package') {
       steps {
          echo 'Build & Deploy'
          sh'''
          docker build -t yte.bilgem.gov.tr/demo_app:v$VERSION .
          docker login -u username -p password yte.bilgem.gov.tr:9092
          docker push yte.bilgem.gov.tr/demo_app:v$VERSION
          '''
       }
    }
    stage('Create git tag') {
       steps {
          withCredentials([gitUsernamePassword(credentialsId: 'hf6783nc-23mb4a-
                 f36-g423-e563454wpfn4', gitToolName: 'Default')]) {
             sh 'git tag -fa $VERSION -m "$VERSION"'
             sh 'git push origin --tags'
          }
       }
    }
 }
 post {
    success {
       script {
          build job:'YTE-SONAR/SONAR-Demo-App' , parameters:[
                  string(name: 'BRANCHNAME',value: getBranch()),
                  string(name: 'VERSION',value:"${VERSION}")
          ] , wait: false
       }
   gitOpsDeploy()
       cleanWs()
    }
    failure {
       cleanWs()
    }
 }
}
```

4.4. Dağıtım (Deploy)

Uygulamaların dağıtımında kubernetes tercih edilmektedir. Kubernetes YAML formatında yazılan nesne tanımlamalarını kullanır. Deployment olarak tanımlanan Kubernetes nesnesi, CD pipeline'ında en aktif nesnedir. Örnekte bir uygulamanın Deployment nesnesi için YAML tanımlamaları vardır. GitOps yaklaşımına göre bu tanımlamalar bir Git deposunda tutulmaktadır. ArgoCD Git deposundaki değişiklikleri otomatik olarak çeker ve Kubernetes'teki çalışan tanımlarla karşılaştırır. Eğer bir değişiklik varsa, bunu bildirim olarak gösterir. ArgoCD üzerinde yapılan ayara göre, çalışan tanımlar güncellenir ve eğer tanımda imaj değişikliği varsa Kubernetes sistemi imajı, imaj deposundan çeker ve yeniden kurar. Yeni konteyner sorunsuz şekilde ayağa kalktıktan sonra eski versiyonlu uygulama konteyneri kapatılır. Uygulama yeni versiyonu ile sunulmuş olur. Uygulamanın sunulması için birden fazla yöntem vardır. Bir yöntemlerden bir tanesi aşağıdaki örnek tanımlaması olası olan Service nesnesi kullanmaktır. Service tanımlaması ile uygulama 8080 portundan dışarı açılmış olur.

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: demo-app
 namespace: default
 labels:
   name: demo-app
spec:
 replicas: 2
 selector:
   matchLabels:
     app: demo-app
 template:
   metadata:
     labels:
       app: demo-app
   spec:
     containers:
       - name: demo-app-pod
         image: yte.bilgem.gov.tr:9092/docker-images/demo-app:v1.0
         ports:
           - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
 namespace: default
 labels:
   visualize: “true”
 name: demo-app-service
spec:
 selector:
   app: demo-app
 ports:
   - name: http
     protocol: TCP
     port: 8080
     targetPort: 8080
 type: NodePort
```

Sonuç ve Öneriler

Bu çalışmada bir uygulamanın CI/CD ardışık süreci uygulanarak kod parçacığı halinden canlı ortamda bir uygulama olarak sunulmasına kadar olan süreç genel hatları ile anlatılmıştır. Bu çözüm bir uygulamayı 10 dakika içinde sıfırdan dağıtılabilecek bir proje haline getirmektedir. Geliştiriciler, geliştirme ve optimizasyon görevlerine odaklanabildikleri için verimleri yükselmiş olur; pipeline'ların oluşturulmasına, yönetilmesine, onarılmasına veya izlenmesine dahil olmaları gerekmez.

Canlı ortama kurulum yapılmadan önce mutlaka bir demo ortamında tüm kontroller yapılmalıdır. Aksi halde büyük veri kayıpları olabilir. Verinin çok kritik olduğu projelerde hatalı veri işlemine sebep olabilir. Güvenlik riskleri oluşturabilir.

CI/CD ilkelerine ilişkin belirsizliği azaltmak, bir proje veya şirkette süreçler düzeyinde daha fazla standardizasyon ve kalite yaratır. CI/CD süreçleri birbirinden ayrılmamalıdır. Doğru tanımlanmış CI/CD sistemi olası bir hatada kurtarma verimliliğini büyük ölçüde artırır ve geliştirmede yer alan kişilerin sayısı ile personel müdahalesi süresini etkili bir şekilde azaltır. Ayrıca, geliştirme ekibi çalışma baskısını hafifletir. Yönetim dönüşümü, bilgi sistemlerinin kapsamlı kalitesini iyileştirir. İşletme ve bakım personelinin profesyonel yeteneği gelişir. İşletme ve bakım yönetiminin teknoloji odaklı dönüşümü büyük başarı sağlar.

Kaynakça

1. https://argo-cd.readthedocs.io/en/stable/
2. https://www.redhat.com/en/topics/devops/what-is-ci-cd
3. https://blog.esper.io/what-is-ci-cd-continuous-integration-and-continuous-delivery-explained/
4. https://kumul.us/understanding-cicd-continuous-integration-deployment-delivery/

TÜBİTAK

BİLGEM

YTE | YAZILIM TEKNOLOJİLERİ ARAŞTIRMA ENSTİTÜSÜ

İşçi Blokları Mahallesi Muhsin Yazıcıoğlu Caddesi No:51/C 06530 Çankaya/ANKARA

+90 (312) 289 92 22 - yte.bilgem@tubitak.gov.tr

TÜBİTAK - BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE)

yte.bilgem.tubitak.gov.tr