WEB PROGRAMLAMA ÖDEVİ PART4

ABDULKADIR OCAL 2024481262

```python
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("http://example.com/login")

username_input = driver.find_element_by_id("username")
password_input = driver.find_element_by_id("password")
submit_button = driver.find_element_by_id("submit")

username_input.send_keys("user123")
password_input.send_keys("pass123")
submit_button.click()

assert "Profile" in driver.title
driver.quit()
```

Bu betik, bir oturum açma işlemini otomatikleştirir ve yönlendirilen sayfanın başlığını doğrulayarak işlemin başarılı olup olmadığını kontrol eder.

DAĞITIM

Verimli Dağıtım Planlaması: Geliştirme aşamasından üretime geçiş, kesintileri en aza indirmek için stratejik planlama gerektirir. Etkili dağıtım planlaması, düşük trafik dönemlerinde dağıtım yapmayı, paydaşlarla önceden iletişim kurmayı ve bir hata durumunda geri alma stratejisi oluşturmayı içerir.

Gelişmiş Dağıtım Araçları: Otomasyon araçları, modern dağıtımlarda önemli bir rol oynar ve doğruluk ile verimliliği sağlar. Jenkins, Docker ve Kubernetes, uygulamaların dağıtımını ve bakımını yönetmek için sıklıkla kullanılır. Aşağıda, dağıtım sürecini otomatikleştiren bir Jenkins pipeline betiği örneği bulunmaktadır:

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/example/project.git'
            }
        }
        stage('Build') {
            steps {
                sh 'make'
            }
        }
        stage('Deployment') {
            steps {
                sh 'scp build/output.jar user@server:/deploy/directory'
            }
        }
    }
}
```

Bu boru hattı en son kodu kontrol eder, projeyi derler ve çıktıyı bir sunucuya dağıtır.

Sürekli İzleme ve Değerlendirme: Dağıtım sonrası, yazılımı dikkatlice izlemek oldukça önemlidir. Bunun için, yazılımın çalışması hakkında gerçek zamanlı içgörüler sunabilen araçlar kullanmak gerekir. İzleme, dağıtım sonrası ortaya çıkan herhangi bir sorunu hızla tespit edip düzeltmeye yardımcı olur ve sistemin kararlılığının korunmasını ve iyi performans göstermesini sağlar.

Zorlukların Üstesinden Gelmek ve En İyi Uygulamalar

Dağıtım Zorluklarını Azaltma: Dağıtım, beklenmeyen kesintiler veya performans düşüşleri gibi sorunlara yol açabilir. Bunları etkili bir şekilde ele almak için:

Dağıtım Öncesi Test: Dağıtımdan önce kapsamlı testler başarısızlık riskini azaltabilir.

Mavi-Yeşil Dağıtım: Bu teknik, iki özdeş üretim ortamını çalıştırarak kesinti süresini ve riski azaltır.

Aşamalı Yük Testi: Yükü kademeli olarak artırmak, sistemi aşırı yüklemeden potansiyel arıza noktalarını belirlemeye yardımcı olabilir.

Sorunsuz Dağıtımların Sağlanması: İyi planlanmış bir dağıtım stratejisi, kapsamlı dokümantasyon ve yeterli eğitimli destek personeli, sorunsuz bir dağıtım için kritik öneme sahiptir. Ayrıca, dağıtım sonrası kullanıcı geri bildirimlerini hızla toplamak ve bunlara göre hareket etmek, uygulamayı ve kullanıcı memnuniyetini iyileştirmeye yardımcı olabilir.

ÇÖZÜM

Sonuç Son test ve dağıtım, yazılım yaşam döngüsünde projenin geliştirmeden gerçek dünya uygulamasına ne kadar iyi geçiş yaptığını belirleyen belirleyici aşamalardır. Kapsamlı testler ekleyerek, dağıtım için otomasyondan yararlanarak ve olası zorluklara hazırlanarak ekipler, yazılımın yalnızca gerekli özellikleri yerine getirmesini değil, aynı zamanda lansman sonrası sağlam ve kullanıcı dostu bir deneyim sunmasını da sağlayabilir.