**"Modern Yazılım Geliştirmede CI/CD'nin Gücünü Keşfetmek: AWS CodePipeline, CodeBuild, CodeDeploy ve CodeCommit ile Uçtan Uca Otomasyon"**

**Hatice Hilal AKSOY**

## PROJE HATIRLATMA

**Google Haberler'den Veri Çekme: AWS Lambda ve S3 Kullanarak Otomatik Haber Toplama**

Ben her saat başı Google haberlerden her kategori altında bulunan haberlerin başlıklarını, link, kısa açıklamalarını, kaynaklarını, yayınlanma tarihlerini ve o anda kaçıncı sırada olduklarını Lamda fonksiyonu kullanarak her saat başı yazılacak şekilde s3 bucket a json formatında kayıt eden bir aws projesi hazırlamıştım. Bu yazıda ise hazırlamış olduğum projeyi nasıl deploy edeceğimi açıklayacağım. İlk başta bu süreçte hangi servisleri kullanacağımızı ve bu servislerin nasıl kullanıldığını anlatacağım ardından benim programım ile bu servislerin entegrasyonunu anlatacağım.

# CI/CD Nedir

CI ve CD, modern geliştirme uygulamalarında ve DevOps'ta sıklıkla karşılaşacağınız terimlerdir. CI/CD, modern yazılım geliştirme uygulamalarının temelini oluşturan iki kavramın kısaltmasıdır: Sürekli Entegrasyon (Continuous Integration - CI) ve Sürekli Dağıtım/Teslimat (Continuous Deployment/Delivery - CD). CI/CD, istediğiniz zaman sürdürülebilir bir şekilde yayınlayabileceğiniz yazılım geliştirme yoludur. Sürekli Entegrasyon (CI) ve Sürekli Teslimat (CD) uygulamalarının birleşik uygulamalarını ifade eder. Sürekli Entegrasyon, CI/CD için bir ön koşuldur ve şunları gerektirir: Geliştiriciler, değişikliklerini günde birçok kez ana kod dalında birleştirir.

```
    Her kod, otomatik bir kod oluşturma ve test sırasını tetiklemek için
birleştirilir.Geliştiriciler ideal olarak sonuçları 10 dakikadan daha kısa sürede
alır, böylece çalışmalarına odaklanmaya devam edebilirler.
```

Sürekli Teslimat uygulamasında ise çoğu durum manuel olarak gerçekleşse de kod değişiklikleri sürekli olarak devreye alınır.

## CI/CD'nin Avantajları

**Verimliliği arttırır**

Otomatik dağıtım ve testler gerçekleştirerek verimliliği artırır. Mühendislik ve operasyon yolu-of aerodinamik ağ üzerinden çalışma imkanı sağlar. Yeni hizmetler ve artan trafik yönetme daha verimli gerçekleşir.

**Azalan riskler**

En son yazılım sürümünü destekleyerek risklere karşı önlem alır. Yapılandırma hata riskini kaldırır. Büyük yükseltme projelerin yerine kademeli güncellemeler sağlar.

## Sürekli Entegrasyon (CI) ve Sürekli Dağıtım/ Teslimat (CD)

### Continuous Integration

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.

Sürekli entegrasyon (CI), geliştiricilerin kod değişikliklerini daha sık geliştirmesine yardımcı olur. Geliştiricilerin bir uygulamadaki değişiklikleri birleştirildiğinde, bu değişiklikler, değişikliklerin uygulamayı bozmadığından emin olmak için uygulamayı otomatik olarak oluşturarak ve farklı düzeylerde testler ile doğrulanmalıdır.

### Continuous delivery

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.

# AWS CodePipeline, CodeBuild, CodeDeploy, ve CodeCommit Tanımları ve Avantajları/Dezavantajları

## CodePipeline Tanımı ve Avantajları/Dezavantajları

Tanımı: AWS CodePipeline, sürekli entegrasyon ve sürekli teslimat (CI/CD) hizmetidir. Kod değişikliklerinin otomatik olarak inşa edilmesini, test edilmesini ve üretim ortamına dağıtılmasını sağlar. Avantajları: Otomasyon: Sürekli entegrasyon ve sürekli dağıtım süreçlerini otomatize eder, böylece yazılım geliştirme süreci daha hızlı ve daha az hata ile gerçekleşir. Esneklik: Farklı geliştirme araçları ve hizmetlerle entegrasyon sağlar, böylece ekipler kendi araçlarını kullanabilirler. Hız: Kod değişikliklerinin hızlı bir şekilde test edilmesini ve üretime alınmasını sağlar, bu da işletmelere pazarda daha hızlı olma avantajı sağlar. Görünürlük: Tüm dağıtım süreci boyunca yüksek düzeyde görünürlük ve izlenebilirlik sağlar. Dezavantajları: Karmaşıklık: Küçük projeler veya basit dağıtım süreçleri için gereğinden fazla karmaşık olabilir. Maliyet: Küçük ekipler veya projeler için maliyetli olabilir, özellikle düşük kullanım durumlarında.

## CodeBuild Tanımı ve Avantajları/Dezavantajları

Tanımı: AWS CodeBuild, tamamen yönetilen bir sürekli entegrasyon hizmetidir. Kaynak kodunu derler, testleri çalıştırır ve hazır yazılımları üretir. Avantajları: Ölçeklenebilirlik: Kaynak kullanımına göre otomatik olarak ölçeklenir, böylece büyük ve küçük projeler için uygundur. Esneklik: Çeşitli programlama dilleri ve yapılandırma seçenekleri ile uyumludur. Entegrasyon: AWS ekosistemiyle ve dış hizmetlerle kolay entegrasyon sağlar. Maliyet Etkinliği: Ödeme yalnızca derleme süresi için yapılır, böylece kaynaklar verimli kullanılır. Dezavantajları: Yapılandırma Zorluğu: İlk kurulum ve yapılandırma, özellikle AWS ile yeni başlayanlar için karmaşık olabilir. Sınırlı Ortam Desteği: Bazı diller veya araçlar için sınırlı destek sunabilir.

## CodeDeploy Tanımı ve Avantajları/Dezavantajları

Tanımı: AWS CodeDeploy, otomatik yazılım dağıtımı sağlayan bir hizmettir. Uygulamaların farklı ortamlara güvenilir, otomatik ve tutarlı bir şekilde dağıtılmasını sağlar. Avantajları: Otomasyon: Dağıtım süreçlerini otomatize ederek zaman ve çaba tasarrufu sağlar. Esneklik: AWS içindeki ve dışındaki sunuculara dağıtım yapabilir. Dayanıklılık: Dağıtım sırasında hataları otomatik olarak işler ve geri alabilir, bu da yüksek kullanılabilirlik sağlar. Ölçeklenebilirlik: Tek bir sunucudan binlercesine kadar ölçeklenebilir. Dezavantajları: Kurulum Karmaşıklığı: İlk kurulum ve yapılandırma bazen zorlayıcı olabilir. Belirli Senaryolar için Sınırlamalar: Tüm dağıtım senaryoları için ideal olmayabilir, bazı özelleştirmeler gerektirebilir.

## CodeCommit Tanımı ve Avantajları/Dezavantajları

Tanımı: AWS CodeCommit, ölçeklenebilir ve güvenli bir şekilde kod depolamak için kullanılan yönetilen bir kaynak kodu kontrol servisidir. Avantajları: Güvenlik: AWS kimlik doğrulama sistemleri ve şifreleme ile yüksek düzeyde güvenlik sunar. Ölçeklenebilirlik: Proje büyüklüğüne ve takımın ihtiyaçlarına göre ölçeklenebilir. Entegrasyon: AWS hizmetleriyle ve CI/CD boru hatlarıyla kolay entegrasyon sağlar. Kullanım Kolaylığı: Git tabanlı bir arayüz sunar, bu da geliştiricilerin mevcut araçlarıyla kolayca çalışmasını sağlar. Dezavantajları: Fonksiyonellik: Bazı gelişmiş özellikler diğer popüler kaynak kodu kontrol sistemlerine göre eksik olabilir. Bağımlılık: AWS ekosistemine güçlü bir bağlılık oluşturur, bu da bazı durumlarda taşınabilirlik sorunlarına yol açabilir.

# GitHub ile AWS CodePipeline Entegrasyonu

GitHub ile AWS CodePipeline entegrasyonu, kod değişikliklerinin otomatik olarak alınıp, test edilip ve AWS üzerinde dağıtılmasını sağlayan bir süreçtir.

1. GitHub'da yeni bir repo oluşturun.

- GitHub'da bir hesabınız yoksa, önce bir hesap oluşturun.
- GitHub'da yeni bir repository oluşturun. Repository oluştururken, repository'nin adını, açıklamasını belirleyebilir ve public/private seçeneğini seçebilirsiniz.
- Yerel bilgisayarınızda, projenizin bulunduğu klasöre gidin.
- Bu klasörde sağ tıklayıp "Git Bash Here" seçeneğini seçerek Git terminalini açın.
- git init komutu ile yeni bir Git repository'si başlatın.

- git config --global user.name "Adınız" ve git config --global user.email "email@adresiniz.com" komutları ile Git kullanıcı adı ve e-posta adresinizi ayarlayın. ya da GitHub'da bir repository'yi klonlamak için, öncelikle GitHub'daki ilgili repository'nin sayfasına gidip "Code" butonuna tıklayarak veya direkt olarak klonlama URL'sini alarak git clone [URL] komutunu kullanabilirsiniz. Örneğin

```
git clone https://github.com/kullaniciAdi/repoAdi.git
```

```
git clone https://git-codecommit.[bölge].amazonaws.com/v1/repos/[repoAdi]
```

2. Dosyaları Commit Etme ve Push Etme

- git add . komutu ile tüm dosyaları staging area'ya ekleyin veya belirli bir dosyayı eklemek için git add dosya_adi komutunu kullanın.

```
git add function.zip
```

- git commit -m "İlk commit" komutu ile değişiklikleri commit edin. Tırnak içindeki mesajı commit'inizle ilgili kısa bir açıklama olarak düzenleyebilirsiniz.

```
git commit -m "first"
```

- git push -u origin master (veya main kullanıyorsanız git push -u origin main) komutu ile yerel değişiklikleri GitHub'daki repository'nize push edin.

```
git push origin main
```

---

## AWS CodePipeline ile GitHub Entegrasyonu

AWS CodePipeline ile GitHub'ı entegre etmek, kod değişikliklerinizin otomatik olarak bir pipeline üzerinden geçirilerek belli bir iş akışı içerisinde test edilmesi, derlenmesi ve dağıtılmasını sağlar. Bu entegrasyon sayesinde, GitHub'daki bir repodan yapılan değişiklikler doğrudan AWS CodePipeline'a aktarılabilir ve sürekli entegrasyon (CI) ve sürekli dağıtım (CD) süreçlerinizi otomatize edebilirsiniz.

1. AWS CodePipeline Oluşturma:

- AWS Management Console'a gidin ve CodePipeline servisine erişin.
- "Create pipeline" seçeneğini seçin.
- Pipeline için bir isim verin ve yeni bir hizmet rolü oluşturun veya var olan bir rolü kullanın.
- "Next" butonuna tıklayarak ilerleyin.

2. Kaynak Kaynağı Olarak GitHub'ı Seçme:

- Kaynak sağlayıcı olarak "GitHub (Version 2)" seçeneğini belirleyin.
- "Connect to GitHub" butonuna tıklayarak AWS'nin GitHub hesabınıza erişmesi için yetki verin.
- İlgili GitHub reposunu ve branch'ı seçin.
- Değişikliklerin nasıl algılanacağını belirleyin (örneğin, her bir GitHub push işlemi için).

3. Derleme Aşaması (Opsiyonel):

- Eğer kodunuzun derlenmesi gerekiyorsa, bir derleme aşaması ekleyebilirsiniz.
- Derleme sağlayıcısı olarak AWS CodeBuild'u seçebilir ve bir build projeleri oluşturabilirsiniz.
- Derleme aşamasında kullanılacak buildspec dosyasını yapılandırabilirsiniz.

4. Dağıtım Aşaması:

- Dağıtım sağlayıcısı olarak AWS CodeDeploy, Amazon S3, Elastic Beanstalk veya başka bir hizmeti seçebilirsiniz.
- Dağıtım aşaması için gerekli yapılandırmaları yapın.

5. Pipeline'ı Başlatma:

- Tüm ayarlar tamamlandıktan sonra, pipeline'ı manuel olarak başlatabilir veya bir GitHub değişikliği tetikleyici olarak kullanabilirsiniz.
- Pipeline, her bir değişiklik için otomatik olarak çalışacak ve kodunuzu belirlediğiniz aşamalardan geçirecektir.

---

## AWS CodePipeline ile CodeCommit

1. Lambda Fonksiyonunu Oluşturma

- AWS Management Console'a gidin ve Lambda'ya erişin.
- "Create function" seçeneğini seçin.
- Fonksiyon için bir isim ve runtime (örneğin, Python 3.8) belirleyin.
- "Create function" butonuna tıklayarak fonksiyonunuzu oluşturun.

2. CodeCommit Repository Oluşturma

- AWS Management Console üzerinden CodeCommit'e gidin.
- "Create repository" seçeneğini tıklayın.
- Repository için bir isim ve açıklama girin.
- "Create" butonuna tıklayarak repository'nizi oluşturun.3. Source Code'u Repository'e Push Etme
- Yerel bilgisayarınızda, Lambda fonksiyonunuz için bir source code oluşturun veya mevcut bir kodu kullanın.
- İlgili CodeCommit repository'sine git ve kodu push edin.

4. CodePipeline Oluşturma

- AWS Management Console üzerinden CodePipeline'a gidin.
- "Create pipeline" seçeneğini tıklayın.
- Pipeline için bir isim girin.
- "New service role" seçeneğini belirleyerek CodePipeline için bir rol oluşturun.
- "Next" butonuna tıklayın.

5. Source Stage Ayarlama

- Source provider olarak "AWS CodeCommit" seçin.
- Oluşturduğunuz repository'yi ve branch'ı seçin.
- "Next" butonuna tıklayın.

6. Build Stage Oluşturma (Opsiyonel)

- Build provider olarak "AWS CodeBuild" seçin.
- Yeni bir build project oluşturun veya mevcut bir projeyi seçin.
- Build konfigürasyonunu ayarlayın ve "Next" butonuna tıklayın. örnek codebuild kodu

```
version: 0.2
phases:
```

```
install:
  runtime-versions:
    python: 3.10
  commands:
    - echo Installing dependencies
    - cd data_fellow
    - pip install -r requirements.txt -t ./
pre_build:
  commands:
    - echo Pre-build phase...
build:
  commands:
    - echo Build started on `date`
    - echo Zipping the function and dependencies...
    - zip -r ../function.zip .
post_build:
commands:
  - echo Post-build phase...
  - aws s3 cp function.zip s3://fellow3/function.zip
```

7. Deploy Stage Ayarlama

- Deploy provider olarak "AWS Lambda" seçin.
- Oluşturduğunuz Lambda fonksiyonunu seçin.
- "Next" butonuna tıklayın ve ayarları gözden geçirin.

8. Pipeline'ı Başlatma

- "Create pipeline" butonuna tıklayarak pipeline'ınızı başlatın.
- CodePipeline, tanımladığınız source'dan kodu alacak, build işlemini gerçekleştirecek (eğer varsa) ve son olarak Lambda fonksiyonunuza deploy edecektir.

9. Değişiklikleri İzleme

- CodeCommit'e yeni bir değişiklik push ettiğinizde, CodePipeline otomatik olarak yeni bir pipeline yürütme işlemi başlatır ve değişiklikleri Lambda fonksiyonunuza uygular.

---

# CodePipline

## AWS CodePipeline ile Lambda Fonksiyonu Otomasyonu

### Kodun Git Yüklenmesi

```
1.lambda fonksiyonunuzun kodunu bir GitHub reposuna yükleyin. Bu, CodePipeline'ın
kodunuzdaki değişiklikleri izlemesini sağlar.
2.Repoya bir buildspec.yml dosyası ekleyin. Bu dosya, CodeBuild tarafından
kullanılacak yapılandırma talimatlarını içerir.
```

### S3 Bucket Oluşturma

Lambda fonksiyonunuzun çıktılarını saklamak için bir tane oluşturun.

```
1.AWS Management Console'a gidin.
2. S3 hizmetine gidin ve "Bucket Oluştur" seçeneğini seçin.
3. Bucket için bir isim ve bölge seçin ve "Oluştur" butonuna tıklayın.
```

### CodePipeline Oluşturma

AWS CodePipeline, kaynak kontrolünden (GitHub) başlayarak, kodunuzun otomatik olarak test edilmesini, derlenmesini ve AWS Lambda'ya dağıtılmasını sağlar.

```
1. AWS Management Console'a gidin ve CodePipeline hizmetine gidin.
2."Pipelines" sekmesine gidin ve "Create pipeline" seçeneğini seçin.
3. Pipeline için bir isim verin ve "Next" butonuna tıklayın.
4."Source provider" olarak "GitHub"ı seçin ve gerekli GitHub kimlik bilgilerini girin.
5. Lambda fonksiyonunuzun bulunduğu repoyu ve branch'i seçin.
6."Build stage" için, "AWS CodeBuild"i seçin ve yeni bir build projesi oluşturun.
7.Bu adım, buildspec.yml dosyanızda tanımlanan talimatları kullanır.
8."Deploy stage" için, "AWS Lambda"yı seçin ve dağıtılacak fonksiyonu belirtin.
9."Create pipeline" butonuna tıklayın.
```

### Pipeline'ı Test Etme

```
1.GitHub reposunda bir değişiklik yapın ve commit edin.
2.CodePipeline otomatik olarak tetiklenecek ve yeni kodu derleyip Lambda
fonksiyonunuza dağıtacaktır.
3.Pipeline'ın başarıyla tamamlandığını ve Lambda fonksiyonunuzun güncellendiğini
doğrulayın.
```

### Lambda Fonksiyonunu Test Etme

```
1.Lambda konsolunda fonksiyonunuzu manuel olarak tetikleyerek yeni kodun beklendiği
gibi çalıştığından emin olun.
2.S3 bucket'ınızı kontrol ederek, Lambda fonksiyonunuzun her saat haberleri başarıyla
çekip çekmediğini doğrulayın.
```

## AWS CloudFormation ile CI/CD Süreci

AWS CodePipeline ile Lambda fonksiyonunuzu otomatik olarak dağıtmak istiyorsanız, CloudFormation şablonunu kullanarak gerekli kaynakları ve dağıtım adımlarını tanımlamanız gerekiyor. Temel olarak, Lambda fonksiyonunuzu, ilgili IAM rollerini, ve diğer gerekli AWS kaynaklarını tanımlayan bir CloudFormation şablonu yazmanız gerekir. Bu şablon, CodePipeline tarafından kullanılacak ve değişiklikler her GitHub'a push yapıldığında otomatik olarak uygulanacak.

İşte süreç:

1. CloudFormation Şablonu Yazma: Lambda fonksiyonunuzu, IAM rollerini, S3 bucket'ınızı ve diğer gerekli kaynakları tanımlayan bir CloudFormation şablonu yazmalısınız. Bu şablon, AWS kaynaklarınızın nasıl oluşturulacağını ve yapılandırılacağını tanımlar.
2. CodePipeline Ayarlama: CodePipeline içinde bir pipeline oluşturursunuz. Bu pipeline, GitHub kaynak kodunuzdan başlayarak, kodunuzu CodeBuild ile build

eden ve ardından CloudFormation kullanarak AWS'de dağıtan adımlardan oluşur.

3. Change Set Kullanma: CloudFormation, yapılan değişiklikleri uygulamadan önce bir "change set" oluşturmanıza olanak tanır. Bu, değişikliklerinizi gözden geçirmenize ve onaylamanıza olanak tanır. CreateChangeSet ve ExecuteChangeSet aksiyonlarını kullanarak bu süreci yönetebilirsiniz.
4. Pipeline'ınızda Adımlar Oluşturma: CodePipeline'da, kaynak kodunuzu alacak, CodeBuild ile build edecek ve ardından CloudFormation şablonunu kullanarak değişiklikleri uygulayacak adımları oluşturmalısınız.
5. CloudFormation Şablonunu Güncelleme: Lambda fonksiyonunuz veya diğer AWS kaynaklarınızda bir değişiklik yapmak istediğinizde, CloudFormation şablonunu güncelleyip GitHub'a push yaparsınız. CodePipeline otomatik olarak yeni değişiklikleri algılar ve dağıtım sürecini başlatır.

örnek cloudformation kodu:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: Lambda function deployment for scraping news and uploading to S3

Resources:
  LambdaExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      Policies:
        - PolicyName: LambdaS3AccessPolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - s3:PutObject
                  - s3:GetObject
                  - s3:ListBucket
                Resource: '*'

  MyLambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      Handler: lambda_function.lambda_handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Runtime: python3.10
      Code:
        S3Bucket: "fellow3"
        S3Key: "function.zip"
```

```
      Timeout: 300  # Adjust the timeout based on your function's requirement

Outputs:
  LambdaFunctionArn:
    Description: "Lambda Function ARN"
    Value: !GetAtt MyLambdaFunction.Arn

  LambdaFunctionName:
    Description: "Lambda Function Name"
    Value: !Ref MyLambdaFunction
```

---

CloudFormation'da "change set" kavramı, bir stack üzerinde yapmayı planladığınız değişiklikleri önceden gözden geçirmenize olanak tanır. Bir change set oluşturduğunuzda, CloudFormation istediğiniz değişiklikleri gerçekleştirmeden önce hangi kaynakların ekleneceğini, değiştirileceğini veya silineceğini gösterir. Bu, özellikle karmaşık altyapı değişikliklerinde, beklenmeyen sonuçları önlemek için çok yararlıdır.

Change set kullanmanın adımları şunlardır:

1. Change Set Oluşturma CloudFormation'a, mevcut stack'inize uygulamak istediğiniz değişiklikleri içeren yeni bir template veya mevcut templatenizdeki değişiklikler gönderirsiniz. CloudFormation, bu değişiklikleri bir change set olarak toplar.
2. Değişiklikleri İnceleme: Change set, hangi kaynakların oluşturulacağını, güncelleneceğini veya silineceğini açıkça gösterir. Bu, değişikliklerinizi uygulamadan önce inceleyebileceğiniz bir "plan" gibidir.
3. Onay: Değişiklikleri inceledikten ve her şeyin beklendiği gibi olduğundan emin olduktan sonra, change set'i yürütebilirsiniz. Bu, değişikliklerin stack'inize uygulanmasını tetikler.

## CodeBuild ve Deploy Aşamaları Arasındaki İlişki

### Lambda ile Deploy ve CodeBuild'in Rolü

CodeBuild Kullanımı: Lambda fonksiyonları için, CodeBuild genellikle fonksiyon kodunu paketler ve bir ZIP dosyası olarak hazırlar. Bu dosya daha sonra Lambda servisi tarafından kullanılabilir. Deploy Aşaması: CodePipeline'da deploy aşamasında, AWS Lambda doğrudan kullanılabilir. Bu durumda, CodeBuild tarafından oluşturulan ZIP dosyası Lambda fonksiyonuna yüklenir ve fonksiyon güncellenir.

### CloudFormation ile Deploy ve CodeBuild'in Rolü

CodeBuild Kullanımı: CloudFormation ile deploy yaparken, CodeBuild genellikle uygulamanın derlenmesi ve gerekli artefaktların hazırlanması için kullanılır. Ayrıca, CloudFormation şablon dosyalarının da hazırlanması veya güncellenmesi bu aşamada gerçekleşebilir. Deploy Aşaması: Deploy aşamasında, oluşturulan artefaktlar ve CloudFormation şablonu kullanılarak altyapı ve uygulama dağıtımı gerçekleştirilir. CloudFormation, şablonda tanımlanan kaynakları oluşturur veya günceller.

### CodeDeploy ile Deploy ve CodeBuild'in Rolü

CodeBuild Kullanımı: CodeDeploy için, CodeBuild uygulamanın derlenmesi ve deploy edilebilir bir paketin (örneğin, bir EC2 instance'ına veya Lambda fonksiyonuna deploy

için gerekli dosyalar) hazırlanması için kullanılır. Deploy Aşaması: CodeDeploy, CodeBuild tarafından hazırlanan artefaktları alır ve bunları belirlenen hedeflere dağıtır. CodeDeploy, canary, mavi/yeşil gibi farklı dağıtım stratejilerini destekler ve uygulamanın belirtilen stratejiye göre dağıtılmasını yönetir.

# AWS CodeDeploy Kullanımı

AWS CodeDeploy, Amazon Web Service'in DevTools ailesinde yer alan ve EC2, Fargate, Lambda gibi şirket içinde çalışan servis hizmetlerine paket dağıtımlarını otomatikleştiren ve tam olarak yönetilen bir dağıtım hizmetidir.AWS CodeDeploy, hızlıca yeni feature'ları canlıya almamızı kolaylaştırır ve uygulama dağıtımı sırasında downtime süresinden kaçınmamıza yardımcı olur ve uygulamalarımızın güncellenmesi sırasında gerekli olan tüm karışık argümanları yönetilen hizmetler kapsamında gerçekleştirmektedir.AWS CodeDeploy ile yazılım dağıtımlarını otomatik hale getirerek aslında manuel olarak gerçekleştirilen işlemleri hatalara açık işlemlere duyulan gereksinimi ortadan kaldırır.Burada bizlere sunulan hizmet aslına bakarsanız, dağıtım gereksinimlerini karşılayacak şekilde ölçeklenmektedir.

## Canary ve Mavi/Yeşil Dağıtım Açıklaması

### Canary

Canary dağıtımı, yeni bir uygulama sürümünü önce sınırlı bir kullanıcı kitlesine sunarak başlar. Bu, "canary" olarak adlandırılan yeni sürümün, geri kalan kullanıcılarınıza genişletilmeden önce gerçek dünya koşullarında test edilmesine olanak tanır. Eğer canary sürümünde herhangi bir sorun tespit edilirse, dağıtım durdurulabilir ve sorun düzeltilene kadar eski sürüme geri dönülebilir. Her şey yolunda giderse, yeni sürüm yavaş yavaş tüm kullanıcılara dağıtılır. Örneğin, yeni bir sürümü önce yalnızca %6'lık bir kullanıcı kitlesine sunabilirsiniz. Bu küçük kullanıcı grubu, yeni sürümdeki potansiyel sorunları erken bir aşamada belirlemenize yardımcı olur. Sorun yoksa, kademeli olarak dağıtımı genişletebilir ve sonunda tüm kullanıcılara ulaştırabilirsiniz.

### Mavi/Yeşil Dağıtım

Mavi/yeşil dağıtımı, iki neredeyse identik üretim ortamından yararlanır: biri "mavi" (mevcut sürüm) ve diğeri "yeşil" (yeni sürüm). Yeşil ortamda yeni sürüm test edilir ve her şey doğru çalıştığında, trafik yavaşça mavi ortamdan yeşil ortama yönlendirilir. Bu strateji, hızlı geri alma imkanı sunar çünkü eğer yeni sürümde bir sorun olursa, trafik hemen mavi ortama geri yönlendirilebilir. Mavi/yeşil dağıtımı, özellikle downtime'ı en aza indirmek ve dağıtım sırasında yüksek kullanılabilirlik sağlamak istediğinizde faydalıdır.

#### AWS CodeDeploy'un Avantajları

Otomatik dağıtımlar Minimum Downtime Sürdürülebilir Entegrasyon

## Proje Entegrasyonu

1. GitHub Reposu Oluşturma ve Kodu Yükleme: GitHub'da bir repo oluşturup Lambda Fonksiyon kodumu yüklediim ve pusk ettim.

2. GitHub ile Entegrasyon: AWS CodePipeline oluşturulurken, kaynak olarak GitHub reposu belirledim. Bu, AWS CodePipeline'ın GitHub webhook'ları aracılığıyla belirli olaylara (örneğin, bir push işlemi) yanıt verebilmesini sağlar. GitHub'da bir değişiklik yapıldığında (örneğin, yeni bir commit push edildiğinde), bu webhook CodePipeline'ı tetikler ve pipeline'ın çalışmasını başlatır.

3. Kaynak Kodunun Alınması: CodePipeline, GitHub'dan kaynak kodunu alır ve bu kodu bir sonraki aşama olan CodeBuild'e geçirir. CodeBuild, çalıştırılacak bir build projesi için yapılandırılmıştır ve bu projenin tanımı bir buildspec.yml dosyasında bulunabilir. Bu dosya, build sürecinin nasıl gerçekleştirileceğini tanımlar.

4. buildspec.yml Dosyası: buildspec.yml, CodeBuild'in build süreci sırasında hangi komutları çalıştıracağını tanımlayan bir YAML dosyasıdır. Bu dosyada tanımlanan adımlar genellikle bağımlılıkların yüklenmesi, testlerin çalıştırılması ve derleme çıktılarının oluşturulması gibi işlemleri içerir. Örneğin, bir Python Lambda fonksiyonu için bağımlılıkların yüklenmesi, testlerin çalıştırılması ve paketleme işlemleri bu dosyada tanımlanabilir.

5. Build Süreci:

- Bağımlılıkların Yüklenmesi: Lambda fonksiyonunuzun gerektirdiği tüm harici Python kütüphaneleri, genellikle bir requirements.txt dosyasında listelenir. buildspec.yml dosyasında, bağımlılıkları yüklemek için pip install -r requirements.txt komutu kullanılır. Bu komut, belirtilen kütüphaneleri ve bağımlılıklarını yükler.
- Unit Testlerin Çalıştırılması: Kaliteli ve hatasız kod sağlamak için unit testler önemlidir. Python'da sıklıkla pytest kullanılır, fakat başka test çatıları da kullanılabilir. Testler, genellikle projenin tests dizininde bulunur. buildspec.yml dosyası içerisinde, pytest tests/ komutu ile tüm testler çalıştırılabilir. Testler başarısız olursa, build süreci durdurulur ve hata giderme için loglar incelenir.
- Lambda Fonksiyonunun Paketlenmesi: Lambda için kod ve bağımlılıkların birlikte paketlenmesi gereklidir. Bu, genellikle bir ZIP dosyası oluşturarak yapılır. Önce, tüm Python bağımlılıkları bir package dizinine yükledim. Bu, pip install -r requirements.txt -t ./package komutu ile yapılabilir. Ardından, Lambda fonksiyonunun kendisi (lambda_function.py gibi) ve package dizini ZIP dosyasına eklenir. Bu, zip -r function.zip ./package lambda_function.py komutu ile gerçekleştirilebilir.
- Artefaktların Hazırlanması: ZIP dosyası oluşturulduktan sonra, bu dosya bir artefakt olarak kabul edilir ve AWS CodeBuild tarafından sonraki deploy aşaması için belirtilen bir konuma (örneğin, bir S3 bucket) yüklenir. buildspec.yml içindeki artifacts bölümü, hangi dosyaların artefakt olarak kabul edileceğini ve nereye yükleneceğini belirtir.

```
version: 0.2
 phases:
  install:
    runtime-versions:
```

```
      python: 3.10
    commands:
      - echo Installing dependencies
      - cd data_fellow
      - pip install -r requirements.txt -t ./
  pre_build:
    commands:
      - echo Pre-build phase...
  build:
    commands:
      - echo Build started on `date`
      - echo Zipping the function and dependencies...
      - zip -r ../function.zip .
post_build:
  commands:
    - echo Post-build phase...
    - aws s3 cp function.zip s3://fellow3/function.zip
```

- install Fazı: runtime-versions: Bu bölüm, build ortamının hangi Python sürümünü kullanacağını belirtir. Burada Python 3.10 sürümü belirlenmiştir. Bu, Lambda fonksiyonunun ve bağımlılıklarının Python 3.10 ile uyumlu olduğunu ve bu sürüm üzerinde çalıştırılacağını gösterir. commands: Bu bölümde, bağımlılıkların nasıl yükleneceği belirtilir. Önce, echo Installing dependencies komutu ile bir log mesajı yazılır. Ardından cd data_fellow komutu ile data_fellow dizinine geçilir. pip install -r requirements.txt -t ./ komutu, requirements.txt dosyasında listelenen Python paketlerini mevcut dizine (./) yükler. -t ./ argümanı, paketlerin bulunduğu dizini belirtir ve burada mevcut dizini işaret eder.
- pre_build Fazı: Bu faz, build aşamasından önceki hazırlıkları içerir. Bu örnekte, echo Pre-build phase... komutu ile sadece bir log mesajı yazılır. Gerçek projelerde bu aşama, yapılandırma dosyalarının ayarlanması, gerekli servislerin başlatılması gibi işlemleri içerebilir.
- build Fazı: Bu faz, asıl build işleminin gerçekleştiği yerdir. echo Build started on \date komutu ile build işleminin başladığı zaman loglanır.echo Zipping the function and dependencies...komutu ile paketleme işleminin başladığı bilgisi loglanır.zip -r ../function.zip . komutu, mevcut dizindeki tüm dosya ve dizinleri (.) rekürsif bir şekilde (-r) bir seviye yukarıdaki dizinde function.zip adıyla bir arşive paketler.
- post_build Fazı: Bu faz, build işlemi tamamlandıktan sonra gerçekleştirilen işlemleri içerir. echo Post-build phase... komutu ile bir log mesajı yazılır. aws s3 cp function.zip s3://fellow3/function.zip komutu, function.zip dosyasını fellow3 adlı S3 bucket'ına kopyalar. Bu komut, AWS CLI'nin s3 cp komutunu kullanarak lokal dosyayı S3'e kopyalar.
- Artefaktların Hazırlanması: ZIP dosyası oluşturulduktan sonra, bu dosya bir artefakt olarak kabul edilir ve AWS CodeBuild tarafından sonraki deploy aşaması için belirtilen bir konuma (örneğin, bir S3 bucket) yüklenir. buildspec.yml içindeki artifacts bölümü, hangi dosyaların artefakt olarak kabul edileceğini ve nereye yükleneceğini belirtir.

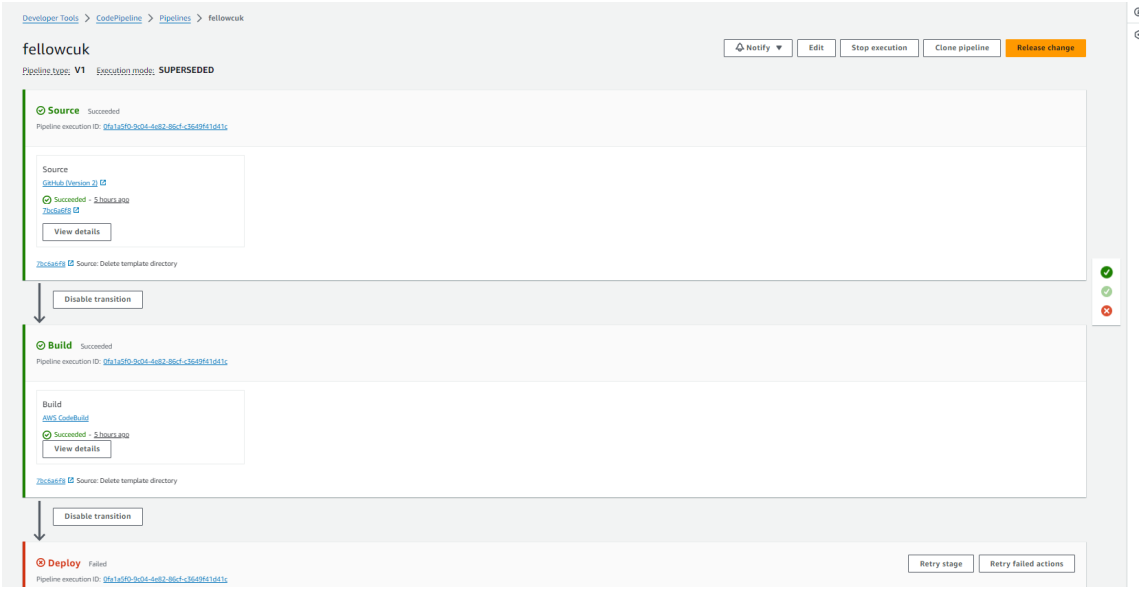
6. AWS CodePipeline Oluşturma: AWS CodePipeline kullanarak bir CI/CD pipeline'ı oluşturun. Kaynak olarak GitHub reposunu kullandım (fellowcuk). AWS CodePipeline, GitHub ile entegrasyon sağlayarak repodaki değişiklikleri otomatik olarak algılayabilir. Build aşaması için AWS CodeBuild projesi ekleyin. CodeBuild, GitHub'dan alınan kaynak kodunu derleyecek ve gerekli testleri çalıştıracaktır.
7. Deploy aşaması: CloudFormation stack'ini güncellemek için bir adım ekleyin. Bu, yeni veya güncellenmiş Lambda fonksiyonunun otomatik olarak dağıtılmasını sağlar. iki adımda deploy sürecini oluşturdum. ilk olarak cloudformation dan yazmış olduğum stack ile create or replace a change set seç ardından sonraki adımda ise execute a change ekledim.
8. Lambda Fonksiyonu ve EventBridge Kuralı: Lambda fonksiyonunuzu, her saat başı Google News'den haber çekip S3 bucket'ına kaydedecek şekilde yazın. EventBridge (CloudWatch Events) kullanarak, her saat başı bu Lambda fonksiyonunu tetikleyecek bir kural oluşturun. CloudFormation stack'inde bu kuralı tanımlayabilirsiniz.
9. S3 Bucket Erişim İzinleri: Lambda fonksiyonunun S3 bucket'ına veri yazabilmesi için gerekli IAM rolü ve izinleri tanımlayın. Bu da CloudFormation tanımlanabilir.

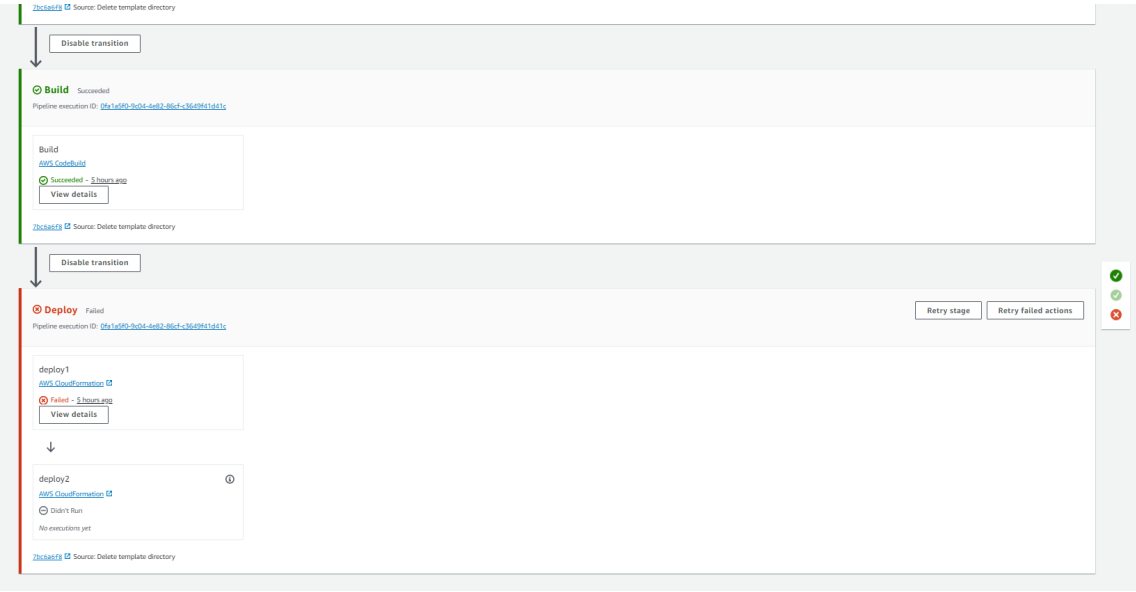
**Sequence Diagram**

---

```
Developer->GitHub: Push Changes
AWS CodePipeline
GitHub->>AWS CodeBuild:
AWS CodeBuild-->AWS CloudFormation:
```

## My CodePipeline






## Karşılaşılan Zorluklar ve Çözümler

GitHub projesini bilgisayara clone ederken zorluk çektim fakat link adımlarını takip ederek çözüme ulaştım. | Hata Çözümü | Stackoverflow |

## Kaynaklar

| AWS Dökumantasyon | AWS Resmi Dokümantasyonu | | AWS CodePipeline | AWS Resmi Dokümantasyonu | | AWS CodeCommit | AWS Resmi Dokümantasyonu | | AWS CodeBuild | AWS Resmi Dokümantasyonu | | AWS CodeDeploy | AWS Resmi Dokümantasyonu |

**Hatice Hilal AKSOY** Linkedin