WEB & YAZILIM GELİŞTİRME SERİSİ · MODÜL 14

Yayınlama

Kodu dünyaya açmak: sürüm kontrolü ve Git iş akışı, dallar ve pull request, sürüm etiketleri, yapı adımı, ortamlar ve sırlar; CI/CD, otomatik test, dağıtım hattı ve stratejileri; geri alma, özellik bayrakları, izleme, loglama, hata takibi; olay yönetimi, yayın güvenliği, sürüm notları ve ekip kültürü. Özgün yayın diyagramlarıyla.

Git · CI/CD · Dağıtım · İzleme · Eğitim amaçlıdır

Bu Kitap Hakkında

Bu modül, bu seride yazmayı öğrendiğin kodu profesyonelce dünyaya açmayı öğretir. Modül 12 bir siteyi bir kez yayına almayı gösterdi; bu modül ise sürekli, güvenli ve tekrarlanabilir yayını — modern geliştirmenin omurgasını — ele alır. Daha çok kavram, iş akışı ve gerçek komutla ilerler; her konuyu net yayın diyagramlarıyla (CI/CD hattı, sürüm çözümlemesi, dağıtım karşılaştırmaları, ortam ve olay şemaları) açıklar. Dört seviye ve yirmi altı bölüm boyunca yayınlama kavramından ve sürüm kontrolünden (Git, dallar, pull request), sürüm etiketlerine (anlamsal sürümleme), yapı adımına, ortamlara (geliştirme/staging/üretim) ve sırların yönetimine; CI/CD'den, sürekli entegrasyona, otomatik testlere, sürekli dağıtıma, dağıtım hattına ve stratejilerine (mavi-yeşil, kanarya, aşamalı); geri almaya, özellik bayraklarına, izleme ve gözlemlenebilirliğe, loglamaya, hata takibine ve sürüm sonrası performansa; olay yönetimine, yayın güvenliğine, dokümantasyona, altyapının kod olmasına (IaC) ve ekip kültürüne kadar uzanır.

Her bölümde konuyu görselleştiren özgün bir diyagram (CI/CD hattı, anlamsal sürüm çözümlemesi, dağıtım stratejisi karşılaştırması, Git akışı, ortam ve olay şemaları), gerçek komutlar ve yapılandırma örnekleri, 'yayında ne olur?' kartı ve bir alıştırma yer alır. Güvenlik ve sorumlu işletim baştan sona vurgulanır: sırların koddan ayrılması, bağımlılık ve sır taraması, en az yetki, loglama ve izlemede veri gizliliği (KVKK), suçlamasız olay analizi. Modül, kodu sık, güvenli ve geri alınabilir biçimde yayınlamanın bütünsel bir kavrayışıyla ve tam bir yayın hattı kontrol listesiyle kapanır. Bu, on altı modüllük 'Web & Yazılım Geliştirme' serisinin on dördüncü modülüdür. Bu seri eğitim amaçlıdır; komut ve sürüm örnekleri temsilîdir.

Web & Yazılım Geliştirme Serisi · Modül 14

İçindekiler

SEVİYE 1

Yayına Hazırlık ve Sürüm Kontrolü

Temeller: yayınlama nedir, sürüm kontrolü ve Git, dallar ve iş akışı, birleştirme ve pull request, sürüm etiketleri, yapı (build) adımı, ortamlar ve yapılandırma/sırlar.

BÖLÜM 01

Yayınlama Nedir?

Yayınlama (deployment), yazdığın kodu çalışan, kullanıcıların erişebildiği bir hâle getirme sürecidir. "Benim makinemde çalışıyor"dan "herkes için yayında ve güvenli"ye geçişin tüm adımlarını kapsar: sürüm kontrolü, yapı, test, dağıtım ve izleme.

Koddan kullanıcıya

Şema 1.1 — Yayınlama döngüsü: sürümle → hazırla → dağıt → izle.

- **Sürüm kontrolü:** her değişiklik kayıtlı ve geri alınabilir (Git).
- **Yapı + test:** kod paketlenir ve otomatik doğrulanır.
- **Dağıtım:** hazır sürüm yayına alınır.
- **İzleme:** yayındaki sistem sürekli gözlenir.

İPUCU

Bu modül, Modül 12'deki (Domain & Hosting) "go-live"in bir adım ötesidir: orada bir siteyi **bir kez** yayına almayı öğrendin; burada **sürekli, güvenli ve tekrarlanabilir** yayınlamayı öğreneceksin. Modern geliştirmede yayınlama nadir ve korkutucu bir olay değil, **sık ve sıradan** bir iştir — günde birçok kez, otomatik ve güvenle yapılır. Bunu mümkün kılan şey **otomasyon ve disiplindir**: sürüm kontrolü (her şey kayıtlı), otomatik test (bozuk kod yakalanır), otomatik dağıtım (insan hatası azalır) ve izleme (sorun anında görülür). Bu seride zaten kod yazmayı öğrendin; bu modül o kodu **profesyonelce dünyaya açmayı** öğretir. Korkmadan, sık sık ve geri alınabilir biçimde yayınlamak, iyi yazılım ekiplerinin imzasıdır.

Yayında ne olur?

YAYIN

İyi kurulmuş bir yayınlama süreciyle, bir geliştirici kodunu gönderdiğinde: değişiklik kaydedilir, otomatik olarak paketlenip test edilir, testler geçerse yayına alınır ve sistem izlenmeye başlar. Tüm bu zincir dakikalar içinde ve çoğu zaman otomatik işler. Kullanıcılar, denenmiş ve çalışan yeni sürümü sorunsuzca kullanmaya başlar; bir aksaklık olursa hızla önceki sürüme dönülür.

Alıştırma

8 dk

Yayınlamayı kavra:

1. Yayınlama sürecinin dört ana adımını kendi cümlelerinle yaz.
2. Modül 12'deki "tek seferlik go-live" ile bu modülün "sürekli yayın" farkını açıkla.
3. Sık ve otomatik yayınlamayı neyin mümkün kıldığını belirt.

BÖLÜM 02

Sürüm Kontrolü ve Git

Yayınlamanın temeli sürüm kontrolüdür: her değişikliğin kim tarafından, ne zaman, neden yapıldığını kaydeden ve geri alınabilir kılan sistem. Git, bunun fiilî standardıdır (Modül 2). Yayınladığın her şey, Git'teki bir kayda dayanır.

Bir değişikliğin yolculuğu

Şema 2.1 — Git: çalışma dizini → hazırlık → kalıcı kayıt (commit).

Temel Git akışı (terminal)

```
 git add .              # değişiklikleri hazırla
git commit -m "Açıklama"   # kalıcı kayıt
git push origin main   # uzak depoya gönder
```

- **Commit:** bir anlık görüntü; geri dönebileceğin bir kayıt.
- **Depo (repo):** projenin tüm geçmişini tutan yer.
- **Uzak depo (remote):** GitHub gibi paylaşılan merkez.

İPUCU

Sürüm kontrolü, yayınlamanın **güvenlik ağıdır**: her commit geri dönebileceğin bir nokta olduğundan, bir şey ters giderse **son sağlam sürüme** dönebilirsin (bu modülün ilerleyen bölümlerindeki "geri alma"nın temeli budur). İyi commit alışkanlıkları yayını kolaylaştırır: **küçük ve sık** commit'ler (her biri tek bir mantıklı değişiklik), **açık mesajlar** (ne ve neden değiştiğini söyleyen) ve değişiklikleri **düzenli olarak uzak depoya göndermek**. Yayınladığın her sürüm bir Git kaydına karşılık gelir — bu yüzden "hangi kod yayında?" sorusunun her zaman net bir yanıtı olur. Modül 2'deki Git temellerin burada doğrudan işe yarıyor; fark, artık Git'i yalnızca kod saklamak için değil, **yayını yönetmek** için kullanman.

Yayında ne olur? YAYIN

Git ile çalıştığında, her commit yayınlanabilir bir "anlık görüntü" oluşturur; uzak depoya (GitHub) gönderdiğinde, bu kayıt ekip ve dağıtım sistemleri için erişilebilir olur. Yayın araçları genelde doğrudan bu depoyu izler: belirli bir dala kod gönderildiğinde dağıtım tetiklenir. Yani Git, hem geçmişin kaydı hem de yayının başlangıç noktasıdır.

Alıştırma 10 dk

Git ile sürümle:

1. Bir commit'in neden "geri dönebileceğin bir nokta" olduğunu açıkla.
2. İyi bir commit mesajının özelliklerini yaz.
3. Uzak depo (remote) ne işe yarar, kendi cümlenle belirt.

BÖLÜM 03

Dallar ve İş Akışı

Dal (branch), ana koddan ayrılıp güvenle değişiklik yapabileceğin paralel bir çalışma alanıdır. Yeni bir özelliği kendi dalında geliştirirsin; ana dal (main) her zaman kararlı ve yayınlanabilir kalır. İş bitince dalını ana dala birleştirirsin.

Özellik dalı iş akışı

Şema 3.1 — Dal iş akışı: main kararlı, geliştirme ayrı dalda.

Dal oluşturmak ve geçmek (terminal)

```
 git checkout -b ozellik/giris-formu   # yeni dal aç + geç
# ... değişiklikleri yap, commit'le ...
git push origin ozellik/giris-formu   # dalı gönder
```

İPUCU

Dal kullanmanın temel kuralı: **main her zaman yayınlanabilir kalmalı**. Yeni bir özellik veya düzeltme üzerinde çalışırken, doğrudan main'de değil, **kendi dalında** çalışırsın — böylece yarım kalmış veya bozuk kodun ana dalı (ve dolayısıyla yayını) etkilemez. Dal isimlerini **anlamlı** tut (`ozellik/giris-formu` , `duzeltme/odeme-hatasi` gibi). Dalları **kısa ömürlü** tut: bir özelliği bitirip hızla birleştir; haftalarca yaşayan dallar, main'den uzaklaştıkça birleştirmesi zorlaşan "çakışmalara" yol açar. Bu "özellik dalı iş akışı" (feature branch workflow), ekiplerin paralel çalışmasını sağlayan en yaygın modeldir: herkes kendi dalında çalışır, hazır olanlar incelemeden geçip main'e katılır. Modül 2'de dalları tanıdın; burada onları **yayın disiplininin** parçası olarak kullanıyorsun.

Yayında ne olur?

YAYIN

Bir özellik dalında çalışırken, yaptığın değişiklikler yalnızca o dalda yaşar; main (ve yayındaki sürüm) sen birleştirene kadar etkilenmez. Bu sayede birden çok kişi aynı projede paralel çalışabilir, herkes kendi işini kararlı ana daldan ayrı geliştirir. Dalın hazır ve incelenmiş olduğunda main'e katılır ve yayına aday hâle gelir.

Alıştırma

10 dk

Dal ile çalış:

1. "main her zaman kararlı kalmalı" ilkesinin neden önemli olduğunu yaz.
2. Bir özellik için anlamlı bir dal adı öner.
3. Dalları neden kısa ömürlü tutmak gerektiğini açıkla.

BÖLÜM 04

Birleştirme ve Pull Request

Pull request (PR), bir dalı ana dala birleştirmeden önce ekibe sunduğun "lütfen incele ve birleştir" isteğidir. PR, kod incelemesinin (code review) merkezidir: değişiklik birleşmeden önce başkaları görür, yorum yapar, onaylar. Bu, kaliteyi ve güveni artırır.

PR akışı

Şema 4.1 — Pull request: sun → otomatik doğrula → incele → birleştir.

- **Kod incelemesi:** başka gözler hata ve iyileştirme yakalar.
- **Otomatik kontroller:** testler PR'da çalışır, geçmezse birleşmez.
- **Tartışma kaydı:** neden böyle yapıldığı PR'da belgelenir.

İPUCU

Pull request, modern ekip geliştirmesinin **kalite kapısıdır**: hiçbir değişiklik, en az bir başka kişinin incelemesinden (ve otomatik testlerden) geçmeden main'e girmez. Bu birçok fayda sağlar: **hatalar erken yakalanır** (iki göz bir gözden iyidir), **bilgi paylaşılır** (ekip kodu tanır), ve **standartlar korunur**. İyi bir PR **küçük ve odaklıdır** — incelemesi kolay olsun diye tek bir işi yapar; devasa PR'lar düzgün incelenemez. İyi bir PR açıklaması **ne** değiştiğini ve **neden** değiştiğini anlatır. İncelerken yapıcı ol: kişiyi değil kodu değerlendir. PR'a bağlı **otomatik kontroller** (bir sonraki seviyedeki CI), testler geçmediğinde birleştirmeyi engelleyerek bozuk kodun main'e sızmasını önler. Bu süreç biraz yavaşlatıyor gibi görünse de, uzun vadede çok daha hızlı ve güvenli yayın sağlar.

Yayında ne olur? YAYIN

Bir PR açtığında, değişikliğin otomatik testlerden geçer ve ekip üyeleri onu inceler; sorun varsa yorum yazar, sen düzeltirsin. Yeterli onay ve geçen testlerle PR birleştirilir ve değişiklik main'e katılarak yayına aday olur. Bu kapı sayesinde main'e yalnızca incelenmiş, test edilmiş kod girer — yayın kalitesi baştan korunur.

Alıştırma 12 dk

PR ile birleştir:

1. Pull request'in iki temel faydasını yaz.
2. Neden büyük PR'lar yerine küçük PR'lar tercih edilir, açıkla.
3. Kod incelerken "kişiyi değil kodu değerlendir" ne demek, belirt.

BÖLÜM 05

Sürüm Etiketleri (Versioning)

Yayınlanan her sürüme bir numara verilir: v2.4.1 gibi. Anlamsal sürümleme (Semantic Versioning), bu numaranın her parçasına net bir anlam yükler. Böylece bir sürümün ne kadar büyük bir değişiklik getirdiği, numarasına bakılarak anlaşılır.

Anlamsal sürümleme

Şema 5.1 — Anlamsal sürümleme: BÜYÜK.KÜÇÜK.YAMA.

- **BÜYÜK (major):** geriye uyumu bozan değişiklik (eski kullanım bozulabilir).
- **KÜÇÜK (minor):** yeni özellik, ama eskiyle uyumlu.
- **YAMA (patch):** yalnızca hata düzeltmesi.

Bir sürümü Git'te etiketlemek

```
git tag -a v2.4.1 -m "Ödeme hatası düzeltildi"
git push origin v2.4.1   # etiketi gönder
```

İPUCU

Anlamsal sürümleme, kullanıcılara ve diğer geliştiricilere **bir bakışta bilgi** verir: numara `1.2.3`'ten `2.0.0`'a sıçradıysa "dikkat, bir şeyler bozulmuş olabilir, güncellemeden önce kontrol et" demektir; `1.2.3`'ten `1.3.0`'a çıktıysa "yeni özellik var ama eskisi çalışmaya devam eder"; `1.2.3`'ten `1.2.4`'e ise "sadece bir hata düzeltildi, güvenle güncelle". Bu ortak dil, özellikle başkalarının kullandığı kütüphaneler ve API'ler için kritiktir. Sürümleri **Git etiketleriyle** (tag) işaretlemek, hangi kodun hangi sürüme karşılık geldiğini kalıcı olarak kaydeder — bir sorun çıktığında "v2.4.0'da çalışıyordu, v2.4.1'de bozuldu" demek ve o noktaya dönmek mümkün olur. Tutarlı sürümleme, profesyonel yayının işaretidir.

Yayında ne olur? YAYIN

Bir sürümü `v2.4.1` olarak etiketleyip yayınladığında, kullanıcılar ve sistemler bu numaradan değişikliğin türünü anlar: BÜYÜK arttıysa dikkatli güncelleme, KÜÇÜK arttıysa yeni özellik, YAMA arttıysa güvenli düzeltme. Git etiketi sayesinde o sürümün tam kodu kalıcı olarak işaretlenir; ileride o noktaya kesinlikle geri dönebilirsin. Sürüm numarası, yayının kimliğidir.

Alıştırma 10 dk

Sürümle:

1. v1.4.2'den sonra şu durumlarda yeni sürüm ne olur: (a) hata düzeltme, (b) yeni uyumlu özellik, (c) uyumu bozan değişiklik?
2. Anlamsal sürümlemenin kullanıcılara ne fayda sağladığını yaz.
3. Git etiketinin (tag) ne işe yaradığını açıkla.

BÖLÜM 06

Yapı (Build) Adımı

Yazdığın kaynak kod, çoğu zaman tarayıcıya veya sunucuya doğrudan gitmez; önce bir "yapı" (build) adımından geçer: derlenir, paketlenir, küçültülür ve yayına uygun bir çıktıya dönüştürülür. Bu adım, geliştirme rahatlığı ile yayın verimliliğini birbirine bağlar.

Kaynaktan dağıtılabilir çıktıya

Şema 6.1 — Yapı adımı: kaynak kodu yayına uygun çıktıya dönüştürür.

Tipik bir yapı komutu

```
npm run build     # kaynağı optimize çıktıya derler
# -> dist/ klasörü oluşur (yayınlanacak dosyalar)
```

- **Derleme/dönüştürme:** modern kod, geniş uyumlu sürüme çevrilir.
- **Paketleme (bundle):** çok dosya, az dosyada birleştirilir.
- **Küçültme (minify):** gereksiz boşluk/isim atılır, dosya küçülür.

İPUCU

Yapı adımı, **geliştirici için okunur kod** ile **kullanıcıya giden hızlı kod** arasındaki köprüdür. Geliştirirken kodu okunur, modüler ve modern yazarsın; build adımı bunu yayına uygun (küçük, hızlı, geniş uyumlu) bir çıktıya dönüştürür. Tipik işler: modern JavaScript'i daha eski tarayıcıların anlayacağı biçime çevirmek, onlarca dosyayı birkaç dosyada birleştirmek (daha az istek = daha hızlı yükleme), ve dosyaları küçültmek (boşlukları/uzun isimleri atmak). Bu yüzden yayınlanan şey, yazdığın kaynak kodun **kendisi değil**, build çıktısıdır — genelde bir `dist/` veya `build/` klasörü. Önemli: build adımının **her ortamda aynı sonucu** üretmesi gerekir (tekrarlanabilirlik); bu yüzden build, sonraki seviyedeki otomatik hatta (CI/CD) dahil edilir. Statik veya basit sitelerde build adımı çok küçük ya da hiç olmayabilir; karmaşık uygulamalarda ise kritik bir aşamadır.

Yayında ne olur? YAYIN

Build komutunu çalıştırdığında, kaynak kodun optimize edilmiş dosyalara dönüşür (örneğin bir `dist/` klasörü); yayınlanan ve kullanıcıya giden şey bu çıktıdır. Build sayesinde site daha hızlı yüklenir ve daha geniş cihaz/tarayıcı yelpazesinde çalışır. Kullanıcı, senin yazdığın ham kodu değil, onun hızlandırılmış ve paketlenmiş hâlini alır.

Alıştırma 10 dk

Build'i kavra:

1. Yapı (build) adımının kaynak koda yaptığı üç işi yaz.
2. Neden yayınlanan şey kaynak kod değil, build çıktısıdır, açıkla.
3. Build'in "her ortamda aynı sonucu üretmesi" neden önemli, belirt.

BÖLÜM 07

Ortamlar: Geliştirme, Test, Üretim

Profesyonel projeler kodu doğrudan kullanıcılara atmaz; kod ortamlardan geçer. Geliştirme ortamında yazarsın, test (staging) ortamında denersin, üretim (production) ortamında kullanıcılar görür. Bu ayrım, hataların kullanıcıya ulaşmadan yakalanmasını sağlar.

Ortam zinciri

Şema 7.1 — Ortamlar: geliştirme → staging → üretim, güvenli terfilerle.

- **Geliştirme:** serbestçe denersin, kırarsın, düzeltirsin.
- **Staging:** üretimin kopyası; son provayı burada yaparsın.
- **Üretim:** gerçek kullanıcıların ve gerçek verinin olduğu ortam.

İPUCU

Ortam ayrımı (Modül 12'de tanıttığımız) yayın güvenliğinin temelidir: bir değişikliği önce **staging**'de (üretimin birebir kopyası) test edip sorun yoksa **üretime** "terfi" ettirirsin. Böylece "doğrudan canlıda deneme" felaketinden kaçınırsın. Kritik bir kural: **ortamlar birbirine karışmamalı** — her ortamın kendi veritabanı, kendi ayarları ve kendi sırları olur. Test verisiyle gerçek kullanıcı verisini asla karıştırma; staging'de yaptığın bir deneme, gerçek müşterileri etkilememeli. Ortamların **mümkün olduğunca birbirine benzemesi** önemlidir (staging üretime ne kadar yakınsa, orada yakalanan hata o kadar gerçekçi olur). Bazı ekipler ek ortamlar da kullanır (QA, ön-üretim), ama temel üçlü — geliştirme, staging, üretim — neredeyse evrenseldir. Bu zincir, kullanıcıların yalnızca denenmiş kodu görmesini sağlar.

Yayında ne olur? YAYIN

Bir değişiklik önce senin makinende (geliştirme) yazılır, sonra üretimin kopyası olan staging ortamında test edilir, sorun yoksa gerçek kullanıcıların bulunduğu üretime alınır. Her ortamın kendi verisi ve ayarı olduğu için denemeler gerçek kullanıcıları etkilemez. Sonuç: kullanıcılar yalnızca staging'de doğrulanmış, çalışan sürümü görür; hatalar onlara ulaşmadan yakalanır.

Alıştırma 10 dk

Ortamları ayır:

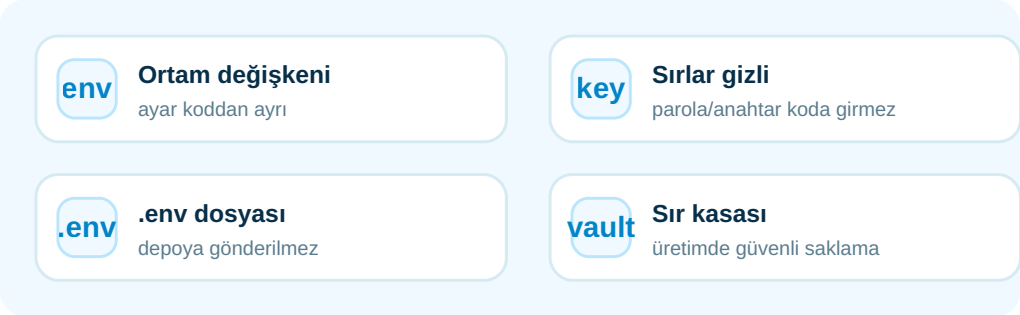
1. Üç ortamı (geliştirme, staging, üretim) ve görevlerini yaz.
2. Ortamların verilerinin neden ayrı tutulması gerektiğini açıkla.
3. Staging'in üretime benzemesi neden önemli, belirt.

BÖLÜM 08

Yapılandırma ve Sırlar

Bir uygulama ortamdan ortama değişen ayarlara ihtiyaç duyar: veritabanı adresi, API anahtarları, parolalar. Bu değerler — özellikle "sırlar" — asla koda gömülmez. Bunun yerine ortam değişkenleri (environment variables) olarak, koddan ayrı tutulur.

Ayarları koddan ayırmak



Şema 8.1 — Yapılandırma: ayarlar ve sırlar koddan ayrı, güvenli tutulur.

Ortam değişkeni: koda gömme, dışarıdan oku

```
 # .env dosyası (depoya GÖNDERİLMEZ, .gitignore'da)
VERITABANI_URL=postgres://...
API_ANAHTARI=gizli-deger

# Kodda: değeri ortamdan oku (gömme!)
const anahtar = process.env.API_ANAHTARI;
```

İPUCU

Bu, tüm serinin güvenlik ilkelerinin yayın tarafındaki en kritik kuralıdır: **sırlar asla koda veya depoya girmez**. Bir API anahtarını veya parolayı koda yazıp Git'e gönderirsen, o sır **geçmişe kalıcı olarak işlenir** ve depoya erişen herkes (ya da depo herkese açıksa tüm dünya) onu görebilir — silsen bile Git geçmişinde kalır. Bunun yerine: ayarları **ortam değişkenleri** olarak tut, yerelde bir `.env` dosyası kullan ve bu dosyayı `.gitignore` ile depodan dışla. Üretimde sırları, hosting sağlayıcının veya bir **"sır kasasının"** (secret manager) güvenli alanında sakla. Aynı kod, her ortamda farklı ayarlarla çalışır — çünkü ayarlar koddan ayrıdır. Bu yaklaşım hem güvenliği sağlar (sır sızmaz) hem esnekliği (ortam değiştirmek için kodu değiştirmen gerekmez). Modül 7-12'deki güvenlik mantığı burada yayın diline çevriliyor: **en az veri, en az açık, sırları koru**.

Yayında ne olur? YAYIN

Ayarları ortam değişkeni olarak tuttuğunda, aynı kod geliştirmede, staging'de ve üretimde farklı değerlerle (farklı veritabanı, farklı anahtar) sorunsuz çalışır — çünkü değerler koda gömülü değil, ortamdan okunur. Sırlar depoya hiç girmediği için sızma riski ortadan kalkar; üretimde güvenli bir kasada saklanır. Sonuç: hem güvenli hem esnek bir yayın yapılandırması.

Alıştırma 12 dk

Sırları koru:

1. Bir API anahtarını neden koda gömmemek gerektiğini açıkla.
2. .env dosyası ve .gitignore birlikte ne işe yarar, yaz.
3. Aynı kodun farklı ortamlarda farklı ayarla çalışmasını ne sağlar, belirt.

SEVİYE 2

Otomatik Dağıtım (CI/CD)

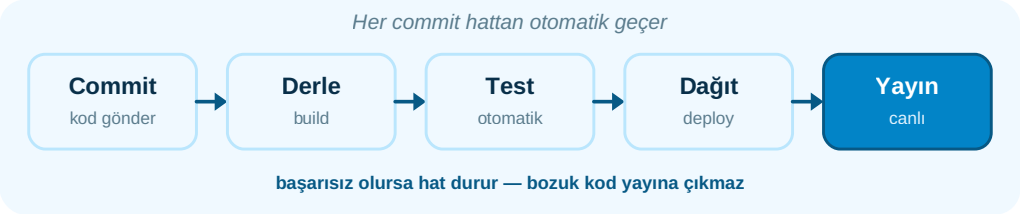
Yayını otomatikleştirmek: CI/CD nedir, sürekli entegrasyon, otomatik testler, sürekli dağıtım, dağıtım hattı (pipeline) ve dağıtım stratejileri.

BÖLÜM 09

CI/CD Nedir?

CI/CD, yayınlamayı elle yapılan riskli bir işten, otomatik ve güvenli bir akışa dönüştüren uygulamadır. CI (Sürekli Entegrasyon) her değişikliği otomatik test eder; CD (Sürekli Dağıtım) test geçen kodu otomatik yayına alır. Birlikte, bir "yayın hattı" oluştururlar.

Otomatik yayın hattı



Şema 9.1 — CI/CD hattı: commit → derle → test → dağıt → yayın, otomatik.

- **CI (Continuous Integration):** her push'ta otomatik derleme + test.
- **CD (Continuous Delivery/Deployment):** geçen kodun otomatik dağıtımı.
- **Hat (pipeline):** bu adımların otomatik zinciri.

Basit bir CI/CD tanımı (kavramsal yapılandırma)

```
# Her commit'te otomatik çalışır:
adimlar:
  - kur:   bağımlılıkları yükle
  - derle: npm run build
  - test:  npm test        # geçmezse hat durur
  - dagit: üretime gönder   # yalnız main dalında
```

İPUCU

CI/CD'nin amacı, yayını **sık, küçük ve güvenli** kılmaktır. Elle yayın yapmak yorucu ve hataya açıktır (bir adımı unutmak, yanlış dosyayı göndermek); CI/CD bu adımları otomatikleştirerek **insan hatasını** en aza indirir ve yayını sıradanlaştırır. **CI** şunu garanti eder: her değişiklik, main'e karışmadan önce otomatik test edilir — bozuk kod erken yakalanır. **CD** ise testten geçen kodu otomatik (veya tek tıkla) yayına taşır. İkisi birden, "ayda bir korkulu yayın" yerine "günde birçok güvenli yayın" sağlar. Anahtar kazanç şudur: küçük ve sık yayınlar, büyük ve seyrek yayınlardan çok daha az risklidir — bir şey bozulursa, neyin bozduğunu bulmak kolaydır (az değişiklik) ve geri almak hızlıdır. CI/CD, modern yazılım ekiplerinin hız ve güveni birlikte elde etmesinin yoludur.

Yayında ne olur? YAYIN

CI/CD kurulduğunda, bir geliştirici kodunu gönderir göndermez hat devreye girer: kod otomatik derlenir, testler çalışır, geçerse staging veya üretime dağıtılır — hepsi insan müdahalesi olmadan, dakikalar içinde. Testler başarısız olursa hat durur ve bozuk kod asla yayına çıkmaz. Sonuç: yayın, korkulan bir olay olmaktan çıkıp güvenilir, tekrarlanan bir rutine dönüşür.

Alıştırma 12 dk

CI/CD'yi kavra:

1. CI ve CD'nin ne yaptığını ayrı ayrı kendi cümlelerinle yaz.
2. Elle yayına göre CI/CD'nin iki avantajını açıkla.
3. "Küçük ve sık yayın neden büyük ve seyrek yayından az risklidir?" yanıtla.

BÖLÜM 10

Sürekli Entegrasyon (CI)

Sürekli entegrasyon, ekip üyelerinin değişikliklerini sık sık (günde birçok kez) ana koda katması ve her katılımın otomatik olarak test edilmesidir. Amaç, sorunları erken ve küçükken yakalamaktır — entegrasyonu sona bırakıp büyük, ağrılı birleştirmeler yaşamak yerine.

Sık entegre et, erken yakala

Şema 10.1 — CI: her değişiklik otomatik test edilir, sorun erken yakalanır.

İPUCU

Sürekli entegrasyonun mantığı şudur: **küçük değişiklikleri sık entegre etmek**, büyük değişiklikleri ara sıra entegre etmekten çok daha kolay ve güvenlidir. Bir geliştirici haftalarca kendi dalında izole çalışıp sonunda devasa bir birleştirme yaparsa, çakışmalar ve sürprizler kaçınılmazdır; bunun yerine günde birkaç kez küçük parçalar entegre edilirse, her adım test edilir ve sorunlar **küçükken** yakalanır. CI'nın kalbi **otomatik testlerdir**: her commit, bir dizi testi otomatik tetikler; testler geçerse kod sağlıklı kabul edilir, geçmezse geliştirici **anında** uyarılır (henüz değişiklik aklındayken). "Build'i bozma" (main'i hep yeşil tut) güçlü bir ekip kültürüdür. CI olmadan, bir hatanın haftalar sonra üretimde keşfedilmesi mümkündür; CI ile aynı hata commit anında yakalanır. Bu, hata düzeltmenin maliyetini dramatik düşürür — geç bulunan hata, erken bulunandan kat kat pahalıdır.

Yayında ne olur?

YAYIN

CI aktifken, bir geliştirici değişikliğini gönderdiği anda sistem otomatik olarak kodu derler ve tüm testleri çalıştırır; sonuç birkaç dakika içinde döner. Testler yeşilse değişiklik güvenle birleştirilebilir; kırmızıysa geliştirici hemen bilgilendirilir ve sorunu, daha karmaşıklaşmadan düzeltir. Böylece ana kod sürekli sağlıklı ve yayınlanabilir kalır.

Alıştırma

10 dk

CI'yı kavra:

1. Sürekli entegrasyonun temel fikrini ("sık entegre et") açıkla.
2. Sorunu erken yakalamanın geç yakalamaya göre avantajını yaz.
3. "Build'i bozma / main'i yeşil tut" ne demek, kendi cümlenle belirt.

BÖLÜM 11

Otomatik Testler

Otomatik testler, kodun beklendiği gibi çalışıp çalışmadığını insan yerine makinenin kontrol etmesidir. CI/CD'nin güvenini sağlayan şey budur: testler olmadan "otomatik yayın", "otomatik felaket" demektir. İyi testler, her yayında kodun hâlâ doğru çalıştığının güvencesidir.

Test türleri

Şema 11.1 — Test türleri: birimden uçtan uca, hepsi otomatik.

Basit bir birim testi (kavramsal)

```
test("topla iki sayıyı toplar", () => {
  beklenen( topla(2, 3) ).esit( 5 );
});
# CI bu testi her commit'te otomatik çalıştırır
```

- **Birim test:** tek bir fonksiyon/parçayı izole dener (hızlı, çok sayıda).
- **Entegrasyon:** parçaların birlikte çalışmasını dener.
- **Uçtan uca (E2E):** kullanıcının gerçek akışını baştan sona dener.

İPUCU

Otomatik testler, CI/CD'nin **güven temelidir**: "testler geçti" demek "kodu güvenle yayınlayabiliriz" demektir. Test yoksa, otomatik dağıtım bozuk kodu doğrudan kullanıcıya gönderebilir — bu yüzden test, otomasyonun olmazsa olmazıdır. Farklı test türleri farklı güven verir: **birim testler** (en çok sayıda, en hızlı) tek tek parçaları doğrular; **entegrasyon testleri** parçaların birlikte çalıştığını; **uçtan uca testler** (en az sayıda, en yavaş) gerçek kullanıcı akışını baştan sona dener. Sağlıklı bir denge "test piramidi" olarak bilinir: çok birim, biraz entegrasyon, az E2E. Testlerin en büyük değeri **regresyonu** (eskiden çalışan bir şeyin yeni değişiklikle bozulması) yakalamasıdır — yeni özellik eklerken eski özelliklerin hâlâ çalıştığından emin olursun. Mükemmel test kapsamı şart değil; **kritik yolları** (giriş, ödeme, ana akışlar) test etmek bile yayın güvenini büyük ölçüde artırır.

Yayında ne olur? YAYIN

Otomatik testler kurulduğunda, her yayın öncesi makineler kodu saniyeler içinde sınar: fonksiyonlar doğru sonuç veriyor mu, parçalar birlikte çalışıyor mu, kullanıcı akışı baştan sona işliyor mu? Bir test başarısız olursa yayın durur ve sorun kullanıcıya ulaşmadan yakalanır. Böylece her yeni sürümde, eski özelliklerin de hâlâ çalıştığından emin olunur — kod büyüdükçe güven azalmaz, artar.

Alıştırma 12 dk

Test et:

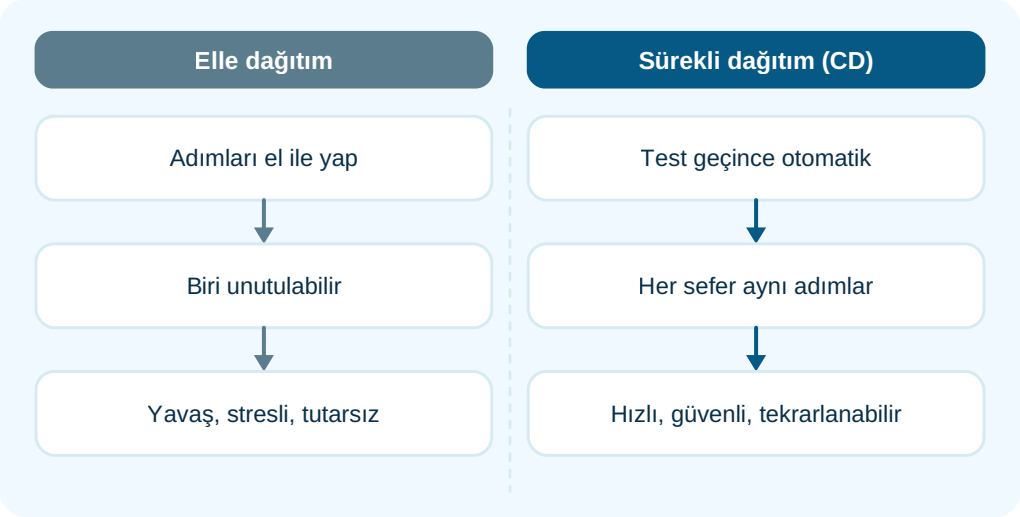
1. Üç test türünü (birim, entegrasyon, uçtan uca) ve farklarını yaz.
2. Otomatik test olmadan otomatik dağıtımın neden riskli olduğunu açıkla.
3. "Regresyon" ne demek ve testler bunu nasıl yakalar, belirt.

BÖLÜM 12

Sürekli Dağıtım (CD)

Sürekli dağıtım, testten geçen kodun otomatik olarak (veya tek bir onayla) yayına alınmasıdır. Elle dağıtımın yorucu, yavaş ve hataya açık adımlarını ortadan kaldırır: kod hazır ve test edilmişse, makine onu güvenle ve tutarlı biçimde yayınlar.

Elle vs otomatik dağıtım



Şema 12.1 — CD: dağıtımı otomatik, tutarlı ve tekrarlanabilir kılar.

- **Continuous Delivery:** her şey hazır, yayın tek onaya bakar.
- **Continuous Deployment:** onay da otomatik (test geçince yayınlanır).
- **Tutarlılık:** her dağıtım aynı adımlarla, aynı sonuçla.

İPUCU

Sürekli dağıtımın iki seviyesi vardır: **Continuous Delivery** (kod her an yayınlanmaya hazırdır, ama son "yayınla" düğmesine bir insan basar) ve **Continuous Deployment** (testler geçer geçmez insan onayı bile olmadan otomatik yayınlanır). Çoğu ekip Delivery ile başlar (kontrolü elde tutmak için) ve güven arttıkça Deployment'a geçer. Her ikisinin de ortak kazancı **tutarlılıktır**: dağıtım her seferinde aynı otomatik adımlarla yapıldığı için, "bir adımı unutmak" veya "ortama göre farklı yapmak" gibi insan hataları ortadan kalkar. Bu, özellikle gece yarısı acil bir düzeltme yaparken paha biçilmezdir — panikle elle adım atmak yerine, denenmiş otomatik hat güvenle çalışır. CD'nin güvenle çalışması için iki şey şarttır: **iyi otomatik testler** (bir önceki bölüm) ve **hızlı geri alma** (bir sonraki seviye) — bir şey ters giderse anında önceki sürüme dönebilmek. Bu üçü birlikte, sık ve korkusuz yayını mümkün kılar.

Yayında ne olur?

YAYIN

CD ile, testten geçen bir değişiklik otomatik olarak (veya tek bir onayla) yayına alınır; dağıtımın her adımı her seferinde aynı biçimde, makine tarafından yapılır. Geliştirici "yayınla" demekten öteye elle bir iş yapmaz — paketleme, taşıma, ayar hepsi otomatiktir. Sonuç: yayın dakikalar sürer, her sefer tutarlıdır ve insan kaynaklı dağıtım hataları neredeyse sıfırlanır.

Alıştırma

10 dk

CD'yi kavra:

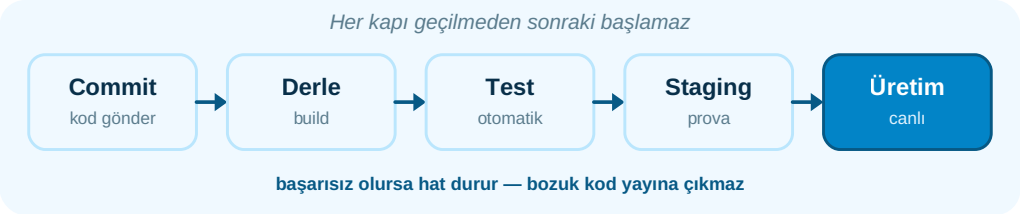
1. Continuous Delivery ile Continuous Deployment farkını yaz.
2. Otomatik dağıtımın "tutarlılık" avantajını bir örnekle açıkla.
3. CD'nin güvenle çalışması için gereken iki şeyi belirt.

BÖLÜM 13

Dağıtım Hattı (Pipeline)

Dağıtım hattı (pipeline), commit'ten yayına kadar geçen tüm otomatik adımların zinciridir. Her adım bir kapıdır: bir önceki başarılı olmadan sonraki başlamaz. Bu, kodun her aşamada doğrulanarak, kontrollü biçimde yayına ilerlemesini sağlar.

Uçtan uca hat



Şema 13.1 — Dağıtım hattı: her aşama bir kontrol kapısıdır.

- **Aşamalı:** derle → test → staging → üretim, sırayla.
- **Kapılar:** bir aşama başarısızsa hat durur.
- **Görünür:** hangi sürümün nerede olduğu her an bellidir.

İPUCU

Dağıtım hattını bir **üretim bandı** gibi düşün: kod, commit'ten yayına kadar bir dizi otomatik istasyondan geçer ve her istasyon bir **kalite kapısıdır** — derleme başarısızsa test başlamaz, test geçmezse staging'e gitmez, staging'de sorun varsa üretime çıkmaz. Bu kademeli yapı, sorunları **doğru aşamada** yakalar ve kötü kodun yayına ilerlemesini engeller. İyi bir hattın özellikleri: **hızlı** (geliştiriciler dakikalar içinde geri bildirim alır — yavaş hat, sık yayını engeller), **güvenilir** (aynı kod hep aynı sonucu verir, "bazen geçer bazen geçmez" testler hattın güvenini öldürür) ve **görünür** (hangi sürümün hangi aşamada olduğu panoda bellidir). Hat, bu modülde öğrendiğin tüm parçaları (sürüm kontrolü, build, test, ortamlar) **tek bir otomatik akışta** birleştirir. Modern yayının kalbi budur: her değişiklik aynı güvenilir borudan geçerek dünyaya ulaşır.

Yayında ne olur? YAYIN

Bir dağıtım hattı kurulduğunda, kod commit'ten yayına kadar otomatik bir zincirden geçer: derlenir, test edilir, staging'e alınıp doğrulanır ve sorun yoksa üretime çıkar. Her aşama bir kontrol kapısı olduğundan, bir adım başarısız olursa hat durur ve sorun o noktada çözülür. Sonuç: yalnızca her kapıdan geçen, baştan sona doğrulanmış kod kullanıcıya ulaşır.

Alıştırma 12 dk

Hattı tasarla:

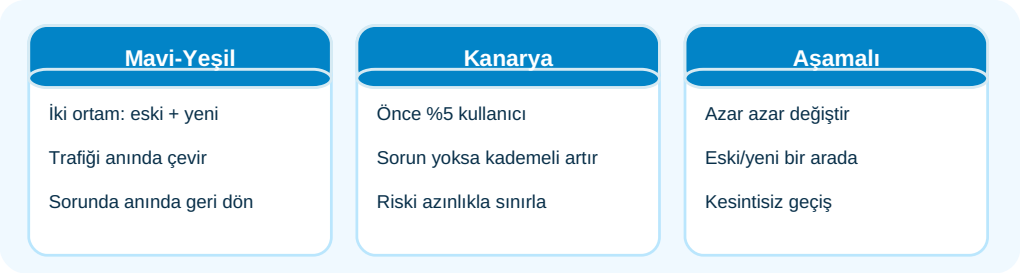
1. Bir dağıtım hattının tipik aşamalarını sırayla yaz.
2. "Her aşama bir kapıdır" ne demek, kendi cümlenle açıkla.
3. İyi bir hattın üç özelliğini (hızlı, güvenilir, görünür) örnekle.

BÖLÜM 14

Dağıtım Stratejileri

Yeni bir sürümü tüm kullanıcılara aynı anda açmak risklidir: bir sorun varsa herkes birden etkilenir. Dağıtım stratejileri, bu riski yönetmenin yollarıdır — yeni sürümü kademeli, geri alınabilir ve kesintisiz biçimde devreye almanı sağlar.

Üç yaygın strateji



Şema 14.1 — Dağıtım stratejileri: riski azaltarak yeni sürüme geçiş.

- **Mavi-Yeşil:** eski (mavi) ve yeni (yeşil) yan yana; trafiği anında çevir, sorunda anında geri al.
- **Kanarya:** yeni sürümü önce küçük bir kullanıcı dilimine aç, sorun yoksa yaygınlaştır.
- **Aşamalı (Rolling):** sunucuları azar azar yeni sürüme geçir, kesinti olmadan.

İPUCU

Bu stratejilerin ortak amacı, yeni sürümün riskini **sınırlamak ve geri alınabilir** kılmaktır. **Mavi-Yeşil**: iki tam ortam tutarsın (mavi=mevcut, yeşil=yeni); yeniyi hazırlayıp test ettikten sonra trafiği bir anda yeşile çevirirsin — sorun çıkarsa trafiği saniyeler içinde maviye geri alırsın (anında, sorunsuz geri dönüş). **Kanarya** (madendeki kanarya benzetmesi): yeni sürümü önce kullanıcıların küçük bir yüzdesine (%1-5) açarsın; bu grupta sorun yoksa kademeli olarak %100'e çıkarırsın — bir hata varsa yalnızca küçük bir grup etkilenir. **Aşamalı/Rolling**: sunucuları teker teker yeni sürüme geçirirsin, böylece hizmet hiç kesilmez. Strateji seçimi altyapına ve riske bağlıdır, ama hepsinin felsefesi aynı: **"hepsini birden değiştirme, kademeli git ve geri dönüş yolunu hazır tut"**. Bu, bir sonraki seviyedeki geri alma ve özellik bayraklarıyla birlikte, korkusuz yayının teknik temelini oluşturur.

Yayında ne olur?

YAYIN

Bir dağıtım stratejisi kullandığında, yeni sürüm tüm kullanıcılara aynı anda değil, kontrollü biçimde açılır: mavi-yeşilde trafiği eski/yeni arasında anında çevirebilir, kanaryada önce küçük bir gruba açıp izleyebilir, aşamalıda sunucuları kesintisiz teker teker güncelleyebilirsin. Bir sorun belirirse etkisi sınırlı kalır ve hızla geri dönülür. Sonuç: kullanıcıların çoğu, riskli geçişten hiç etkilenmeden yeni sürüme kavuşur.

Alıştırma

12 dk

Strateji seç:

1. Üç dağıtım stratejisini kendi cümlelerinle özetle.
2. Kanarya dağıtımın "riski sınırlama" mantığını açıkla.
3. Bu stratejilerin ortak felsefesini ("kademeli + geri alınabilir") yaz.

SEVİYE 3

Güvenli ve Sürdürülebilir Yayın

Yayını güvenli ve sağlıklı tutmak: geri alma, özellik bayrakları, izleme ve gözlemlenebilirlik, loglama, hata takibi ve sürüm sonrası performans.

BÖLÜM 15

Geri Alma (Rollback)

En iyi testlere rağmen bazen bozuk bir sürüm yayına çıkar. O anda en önemli yetenek, hızlıca önceki sağlam sürüme dönebilmektir: geri alma (rollback). İyi bir yayın sistemi, "ileri gitmeyi" kadar "hızlı geri dönmeyi" de kolay kılar.

Sorunda hızla geri dön

Şema 15.1 — Geri alma: önce eski sürüme dön, sonra sakince düzelt.

Bir değişikliği geri almak (Git)

```
 git revert    # bozan değişikliği tersine çevir
# veya: önceki sürüm etiketine dağıt
#   deploy v2.4.0     (v2.4.1 bozulduysa)
```

İPUCU

Geri alma yeteneği, korkusuz yayının **güvenlik ağıdır**: bir sürümün bozuk çıkması felaket değildir — **hızla geri alınamaması** felakettir. Bu yüzden iyi ekipler, daha bir sürümü yayınlamadan önce "geri dönüş planını" hazır tutar. Altın kural: **önce düzeltme değil, önce geri alma**. Üretimde bir şey bozulduğunda, panikle canlıda düzeltmeye çalışmak yerine, önce sistemi son sağlam sürüme döndürürsün (kullanıcılar anında kurtulur), sonra sorunu sakince, baskı olmadan staging'de incelersin. Mavi-yeşil dağıtım (önceki bölüm) geri almayı anlık yapar (trafiği maviye çevir); Git ile `revert` veya önceki sürüm etiketine yeniden dağıtım da yaygın yöntemlerdir. **Dikkat:** veritabanı değişiklikleri geri almayı zorlaştırır — bir alanı silen bir değişikliği geri almak zordur; bu yüzden veri değişikliklerini **geriye uyumlu** ve dikkatli planlamak gerekir. Hızlı, denenmiş bir geri alma yolu, ekibin sık ve cesur yayın yapmasını mümkün kılar.

Yayında ne olur? YAYIN

Bir sürüm sorun çıkardığında, geri alma ile sistem saniyeler/dakikalar içinde önceki sağlam sürüme döner; kullanıcılar bozuk sürümle uzun süre uğraşmaz, hizmet hızla normale döner. Ardından ekip, baskı altında olmadan hatayı bulup düzeltir ve düzeltilmiş sürümü yeniden yayınlar. Sonuç: bir yayın hatası, saatlerce süren bir kriz yerine, birkaç dakikalık yönetilebilir bir aksaklığa dönüşür.

Alıştırma 10 dk

Geri al:

1. "Önce geri al, sonra düzelt" ilkesinin neden doğru olduğunu açıkla.
2. Mavi-yeşil dağıtımın geri almayı nasıl kolaylaştırdığını yaz.
3. Veritabanı değişikliklerinin geri almayı neden zorlaştırdığını belirt.

BÖLÜM 16

Özellik Bayrakları (Feature Flags)

Özellik bayrağı, bir kod parçasını yayına almakla onu kullanıcıya açmayı birbirinden ayırır. Kodu üretime gönderirsin ama özellik "kapalı" durur; hazır olduğunda bir anahtarla açarsın — kod yeniden yayınlamadan. Bu, yayın ile sürüm yayımını ayıran güçlü bir tekniktir.

Yayınla ama kontrollü aç

Şema 16.1 — Özellik bayrağı: yayınla, sonra kontrollü aç/kapat.

İPUCU

Özellik bayrakları, **"yayınlamak"** (kodu üretime koymak) ile **"yayımlamak"** (özelliği kullanıcıya açmak) arasındaki bağı koparır — bu çok güçlü bir esnekliktir. Avantajları: **kademeli açma** (önce çalışanlara, sonra %5, sonra herkese — kanarya gibi ama kod düzeyinde), **anında kapatma** (bir özellik sorun çıkarırsa, yeniden dağıtım beklemeden bayrağı kapatıp anında geri alırsın), **yarım işi güvenle birleştirme** (tamamlanmamış bir özelliği bayrak arkasında main'e katabilirsin, kullanıcı görmez — uzun ömürlü dalları önler), ve **denemeler** (farklı kullanıcılara farklı sürüm gösterip karşılaştırma). Dikkat edilecekler: bayraklar **karmaşıklık** ekler (çok sayıda açık bayrak kodu takip edilemez kılar) — bu yüzden işi biten bayrakları **temizle**. Ayrıca bayrak mantığını ve hangi kullanıcının ne gördüğünü **gizlilik** ilkelerine uygun yönet. Geri alma ve dağıtım stratejileriyle birlikte, özellik bayrakları "yayını olaydan rutine çeviren" araç setini tamamlar.

Yayında ne olur? YAYIN

Özellik bayrağı kullandığında, yeni bir özelliğin kodu üretimde durur ama kapalıdır; kullanıcılar onu görmez. Hazır olduğunda bayrağı önce küçük bir gruba, sonra herkese açarsın — kodu yeniden yayınlamadan, anında. Bir sorun çıkarsa bayrağı kapatman yeterlidir; özellik anında devre dışı kalır, kullanıcılar korunur. Böylece yayın (kod gönderme) ve yayımlama (özelliği açma) birbirinden bağımsız, kontrollü iki işe dönüşür.

Alıştırma 12 dk

Bayrak kullan:

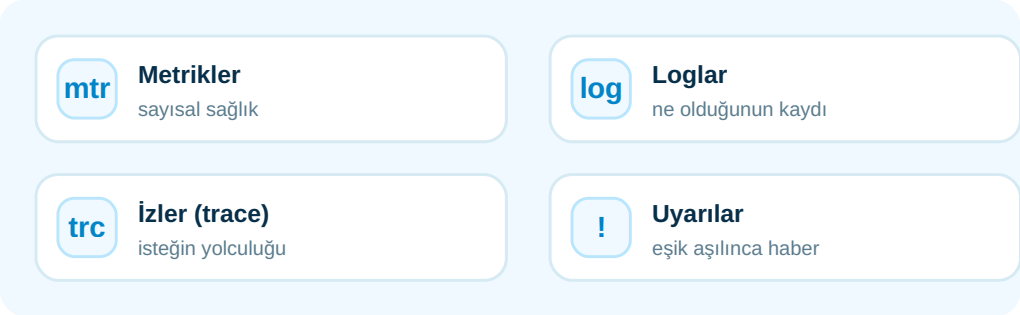
1. "Yayınlamak" ile "yayımlamak" arasındaki farkı özellik bayrağı bağlamında açıkla.
2. Özellik bayrağının iki avantajını yaz.
3. Bayrakların eklediği karmaşıklığı yönetmek için ne yapılmalı, belirt.

BÖLÜM 17

İzleme ve Gözlemlenebilirlik

Yayına almak işin yarısıdır; diğer yarısı yayındaki sistemin sağlıklı olduğundan emin olmaktır. İzleme (monitoring) ve gözlemlenebilirlik (observability), sistemin içinde ne olup bittiğini sürekli görmeni sağlar — bir sorun, kullanıcılar şikâyet etmeden önce fark edilir.

Sistemin sağlığını gör



Şema 17.1 — Gözlemlenebilirliğin üç direği: metrik, log, iz + uyarılar.

- **Metrikler:** yanıt süresi, hata oranı, kaynak kullanımı gibi sayılar.
- **Loglar:** olayların zaman damgalı kaydı (bir sonraki bölüm).
- **İzler (traces):** bir isteğin sistemdeki tüm yolculuğu.
- **Uyarılar:** bir eşik aşıldığında otomatik bildirim.

İPUCU

İzlemenin amacı, sistemin sağlığını **kullanıcıların gözünden önce** görmektir. "Gözlemlenebilirlik" üç direğe dayanır: **metrikler** (sayısal göstergeler — yanıt süresi, dakikadaki hata sayısı, CPU/bellek; "ne kadar iyi/kötü" sorusunu yanıtlar), **loglar** (olayların metin kaydı; "tam olarak ne oldu" sorusunu yanıtlar — sonraki bölüm), ve **izler/traces** (bir isteğin sistemin parçaları arasındaki yolculuğu; "yavaşlık/hata nerede" sorusunu yanıtlar). Bunların üstüne **uyarılar** kurulur: bir metrik tehlikeli bir eşiği aştığında (örneğin hata oranı %1'i geçtiğinde) ekip otomatik bilgilendirilir. İyi izlemenin işareti, bir sorunu **müşteri aramadan önce** bilmektir. Aşırıya kaçma tuzağına dikkat: çok fazla ve gereksiz uyarı "uyarı yorgunluğu" yaratır ve gerçek uyarılar kaçırılır — yalnızca **eyleme geçmeyi gerektiren** şeyler için uyarı kur. İzleme, bu modülün baştan beri vurguladığı şeyi tamamlar: yayın, "gönder ve unut" değil, "gönder ve gözle"dir.

Yayında ne olur? YAYIN

İzleme kurulduğunda, sistemin sağlığı sürekli ölçülür: yanıt süreleri, hata oranları, kaynak kullanımı bir panoda görünür ve bir değer tehlikeli eşiği aştığında ekip otomatik uyarılır. Loglar ve izler, bir sorun çıktığında tam olarak nerede ve neden olduğunu gösterir. Sonuç: sorunlar kullanıcılar fark etmeden yakalanır, kesintiler kısalır ve sistemin gerçek davranışı tahminle değil, veriyle bilinir.

Alıştırma 10 dk

İzle:

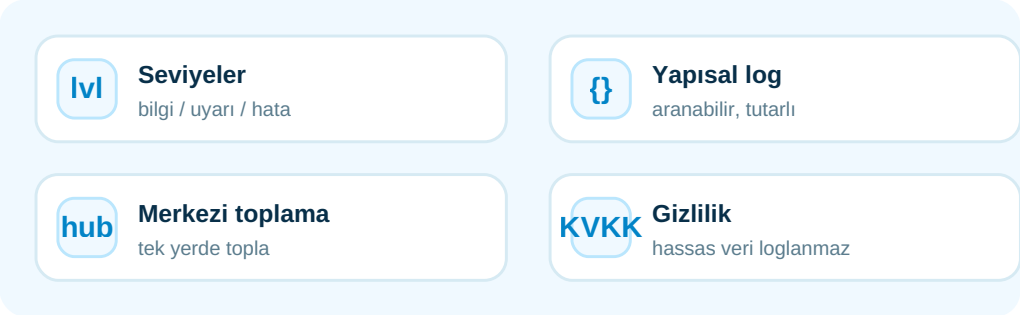
1. Gözlemlenebilirliğin üç direğini (metrik, log, iz) ve hangi soruyu yanıtladıklarını yaz.
2. İyi izlemenin "müşteriden önce bilmek" hedefini açıkla.
3. "Uyarı yorgunluğu" nedir ve nasıl önlenir, belirt.

BÖLÜM 18

Loglama (Günlükleme)

Loglar, sistemin yaşadığı olayların zaman damgalı kaydıdır: ne oldu, ne zaman, hangi koşulda. Bir sorun çıktığında, "kara kutu" gibi geriye dönüp ne olduğunu anlamanı sağlayan birincil araçtır. Ama loglama dikkat ister — özellikle gizlilik açısından.

Olayların kaydı



Şema 18.1 — Loglama: seviyeli, yapısal, merkezi ve gizliliğe saygılı.

- **Seviyeler:** bilgi (info), uyarı (warning), hata (error) — önemi ayır.
- **Yapısal:** tutarlı biçimde yaz (aranabilir, makinece işlenir).
- **Merkezi:** tüm sunucuların logları tek yerde toplanır.

İPUCU

Loglar bir sorunu çözerken en çok başvuracağın kaynaktır: "kullanıcı şu saatte hata aldı" denildiğinde, o ana ait loglar ne olduğunu anlatır. İyi loglama için: **seviyeleri kullan** (rutin bilgiler "info", dikkat çekenler "warning", gerçek sorunlar "error" — böylece gürültüden sinyali ayırırsın); **yapısal yaz** (serbest metin yerine tutarlı, aranabilir biçim — sorun anında ilgili logu hızla bulursun); **merkezi topla** (onlarca sunucunun loguna tek tek bakmak yerine hepsini tek bir yerde ara). Ama en kritik kural **gizliliktir**: **hassas veriyi asla loglama** — parolalar, kart numaraları, kişisel veriler (KVKK kapsamındakiler) loglara yazılırsa, loglar bir veri sızıntısı kaynağına dönüşür. Logları da bir veri olarak gör: kim erişebilir, ne kadar saklanır, KVKK'ya uygun mu? "Her şeyi logla" değil, **"doğru şeyi, gizliliğe saygıyla logla"** ilkesini benimse. Loglar olmadan üretimdeki bir sorunu çözmek karanlıkta el yordamıdır; iyi loglar ışık tutar.

Yayında ne olur? YAYIN

Loglama kurulduğunda, sistemde olan önemli olaylar (istekler, hatalar, uyarılar) zaman damgasıyla kaydedilir ve merkezi bir yerde toplanır. Bir sorun çıktığında, ilgili zaman aralığının loglarına bakarak ne olduğunu adım adım yeniden kurabilirsin — tahmin etmek yerine görürsün. Hassas veriler loglanmadığı için bu kayıtlar bir gizlilik riski yaratmaz. Sonuç: sorunlar daha hızlı teşhis edilir, sistem davranışı şeffaf olur.

Alıştırma 10 dk

Logla:

1. Log seviyelerinin (info/warning/error) ne işe yaradığını yaz.
2. Yapısal ve merkezi loglamanın avantajını açıkla.
3. Hangi verilerin asla loglanmaması gerektiğini ve nedenini (KVKK) belirt.

BÖLÜM 19

Hata Takibi ve Uyarılar

Üretimde hatalar kaçınılmazdır; önemli olan onları hızla fark edip düzeltmektir. Hata takibi (error tracking), kullanıcıların yaşadığı hataları otomatik yakalar, gruplar ve ekibe bildirir — böylece bir hata, sessizce kullanıcıları rahatsız etmek yerine hemen görülür ve çözülür.

Hatayı yakala, ekibi uyar

Şema 19.1 — Hata takibi: otomatik yakala, ekibi uyar, hızla düzelt.

İPUCU

Hata takibinin en önemli gerçeği şudur: **kullanıcıların çoğu, yaşadığı hatayı bildirmez** — sessizce ayrılır. Bu yüzden hataları onların bildirmesine güvenemezsin; **otomatik yakalaman** gerekir. Bir hata takip sistemi, üretimde oluşan her hatayı (mesajı, oluştuğu yeri, kaç kullanıcıyı etkilediğini, hangi koşulda olduğunu) otomatik kaydeder, aynı hataları **gruplar** (bin kez olan aynı hata, bin ayrı bildirim değil tek bir grup olur) ve önemine göre ekibi **uyarır**. Bu, "hangi sorunu önce çözmeliyim?" sorusunu yanıtlar — en çok kullanıcıyı etkileyen hataya öncelik verirsin. **Uyarı eşiklerini akıllıca** ayarla: her küçük hata için herkesi uyandırma (uyarı yorgunluğu), ama kullanıcıyı gerçekten etkileyen sorunlar anında bilinmeli. Hata takibi, izleme ve loglama ile birlikte yayın sonrası "gözler"i tamamlar: yayınladıktan sonra sistemin gerçek dünyada nasıl davrandığını görür, sorunları hızla yakalar ve sürekli iyileştirirsin. Hataları gizlemek değil, **hızla görüp düzeltmek** olgun bir yayın kültürünün işaretidir.

Yayında ne olur?

YAYIN

Hata takibi kurulduğunda, üretimde bir kullanıcı hata yaşadığında bu otomatik olarak detayıyla kaydedilir ve ekip bilgilendirilir — kullanıcının şikâyet etmesini beklemeden. Aynı hatalar gruplanır ve etkilerine göre önceliklendirilir, böylece ekip en çok kullanıcıyı etkileyen sorunu önce çözer. Sonuç: hatalar sessizce birikmek yerine hızla yüzeye çıkar ve düzeltilir; ürün her yayında biraz daha sağlamlaşır.

Alıştırma

10 dk

Hataları yakala:

1. Kullanıcıların hataları neden bildirmediğini ve bunun sonucunu yaz.
2. Hata takibinin "gruplama ve önceliklendirme" faydasını açıkla.
3. Uyarı eşiklerini akıllıca ayarlamanın neden önemli olduğunu belirt.

BÖLÜM 20

Performans ve Sürüm Sonrası

Bir sürüm yayınlandıktan sonra iş bitmez: gerçek dünyada nasıl davrandığını izlemek gerekir. Yayın öncesi her şey "yolunda görünebilir" ama gerçek kullanıcılar, gerçek yük ve gerçek cihazlar sürprizler çıkarır. Sürüm sonrası izleme, varsayımı gerçekle değiştirir.

Varsayım değil, gerçek veri

Şema 20.1 — Sürüm sonrası: gerçek kullanıcı verisiyle doğrula.

- **Yanıt süresi:** sayfalar/işlemler gerçekte ne kadar sürüyor?
- **Hata oranı:** yeni sürümle hatalar arttı mı?
- **Kaynak kullanımı:** sunucu yükü sürdürülebilir mi?
- **Kullanıcı etkisi:** davranış beklendiği gibi mi?

İPUCU

"Bende çalışıyordu" ile "üretimde çalışıyor" arasında büyük bir fark vardır: gerçek dünya, sınırlı test ortamının öngöremediği koşullar (gerçek trafik hacmi, çeşitli cihazlar, beklenmedik kullanıcı davranışları, kenar durumlar) sunar. Bu yüzden bir sürümü yayınladıktan sonra onu **aktif olarak izlemek** şarttır — özellikle ilk saatlerde. Bakacağın temel göstergeler: **yanıt süresi** (yavaşladı mı?), **hata oranı** (yeni sürümle hatalar arttı mı?), **kaynak kullanımı** (sunucular zorlanıyor mu?) ve **kullanıcı davranışı** (yeni özellik beklendiği gibi kullanılıyor mu, yoksa insanlar takılıyor mu?). Yeni sürümün göstergelerini **önceki sürümle karşılaştır** — ani bir kötüleşme, bir sorunun işaretidir ve geri alma gerekebilir. Bu, yayını bir "olay" olmaktan çıkarıp **sürekli bir öğrenme döngüsüne** dönüştürür: yayınla, izle, öğren, iyileştir, tekrar yayınla. Veriyle yönetilen bu yaklaşım, varsayıma dayalı tahminden çok daha sağlamdır — ve bir sonraki yayını daha iyi yapmanı sağlar.

Yayında ne olur? YAYIN

Bir sürümü yayınladıktan sonra izlediğinde, onun gerçek dünyada nasıl davrandığını görürsün: sayfalar gerçekten hızlı mı, hatalar arttı mı, sunucular yükü kaldırıyor mu, kullanıcılar yeni özelliği beklendiği gibi mi kullanıyor? Bu veriler, yayın öncesi varsayımlarını doğrular veya çürütür; bir kötüleşme görürsen hızla müdahale eder (gerekirse geri alırsın). Sonuç: her yayın, bir sonrakini daha iyi yapan bir öğrenme fırsatına dönüşür.

Alıştırma 10 dk

Sürüm sonrasını izle:

1. "Bende çalışıyordu" ile "üretimde çalışıyor" farkını bir örnekle açıkla.
2. Yayın sonrası izlenecek dört temel göstergeyi yaz.
3. Yeni sürümü önceki sürümle karşılaştırmanın neden önemli olduğunu belirt.

SEVİYE 4

İşletim ve Olgunluk

Yayını işletmek ve olgunlaştırmak: olay yönetimi, yayın güvenliği, dokümantasyon ve sürüm notları, otomasyon ve altyapı (IaC), ekip kültürü ve yayın hattı kontrol listesi.

BÖLÜM 21

Olay Yönetimi (Incident)

Olay (incident), kullanıcıları etkileyen ciddi bir sorundur: site çöktü, ödeme çalışmıyor, veri erişilemez. Ne kadar iyi hazırlanırsan hazırlan, olaylar yaşanır. Önemli olan, panik yerine sakin ve sistematik bir müdahale sürecine sahip olmaktır.

Olaya müdahale akışı

Şema 21.1 — Olay yönetimi: tespit → müdahale → çöz → ders çıkar.

İPUCU

İyi olay yönetimi, kaosu **sürece** çevirir. Adımlar: **tespit** (izleme ve uyarılar olayı kullanıcılardan önce yakalar — bu yüzden Seviye 3 bu kadar önemli); **müdahale** (ilk hedef sorunu "anlamak" değil, **hizmeti kurtarmaktır** — genelde hızlı geri alma; tıbbi acil gibi, önce kanamayı durdur); **çözüm** (sistem stabil olunca kalıcı düzeltmeyi sakince yap ve doğrula); ve en önemlisi **ders çıkarma** (olay sonrası "post-mortem"). Bu son adımın kritik kuralı **suçlamasızlıktır** (blameless): amaç "kimin hatası?" değil, "sürecin neresi bu hataya izin verdi ve nasıl önleriz?" sorusudur. Suçlama kültürü, insanları hataları gizlemeye iter ve aynı hatalar tekrarlanır; suçlamasız kültür ise açık konuşmayı ve gerçek iyileştirmeyi sağlar. Olaylara hazırlık için: net bir **müdahale planı**, kimin ne yapacağının bilindiği **roller**, ve sakin **iletişim** (kullanıcıları durum sayfasıyla bilgilendirmek). Olgun ekipler olayları kaçınılması gereken utançlar değil, **sistemi güçlendiren öğrenme fırsatları** olarak görür.

Yayında ne olur?

YAYIN

Bir olay yaşandığında, iyi bir süreç şöyle işler: izleme sorunu tespit eder, ekip önce hizmeti kurtarır (genelde geri alarak), sonra kalıcı düzeltmeyi sakince yapar ve son olarak suçlama olmadan ne olduğunu analiz edip dersleri kayda geçirir. Bu disiplin, krizleri kısaltır ve aynı hatanın tekrarını önler. Sonuç: her olay, sistemi ve süreci biraz daha sağlamlaştıran bir öğrenmeye dönüşür.

Alıştırma

12 dk

Olayı yönet:

1. Olaya müdahalenin dört adımını sırala.
2. "Önce hizmeti kurtar, sonra anla" ilkesini açıkla.
3. "Suçlamasız analiz" (blameless) neden önemli, kendi cümlenle yaz.

BÖLÜM 22

Yayın Güvenliği

Yayınlama süreci, güvenliğin de bir parçasıdır: hangi kodun, hangi bağımlılıkların ve hangi sırların yayına gittiği dikkatle yönetilmelidir. Modern saldırıların çoğu, koddaki bir açıktan değil, güvenilmeyen bir bağımlılıktan veya sızan bir sırdan gelir.

Yayın hattını güvenli tut

Şema 22.1 — Yayın güvenliğinin dört temel önlemi.

- **Bağımlılık taraması:** kullandığın paketlerdeki bilinen açıkları otomatik kontrol et.
- **Sır taraması:** koda yanlışlıkla sızmış anahtar/parola var mı, tara.
- **En az yetki:** dağıtım hattı yalnızca gereken erişime sahip olsun.

İPUCU

Yayın güvenliği, tüm serinin güvenlik ilkelerini yayın hattına taşır. Üç büyük risk alanı: **(1) Bağımlılıklar** — modern projeler onlarca dış paket kullanır; bunlardan birinde bilinen bir açık varsa, senin projen de savunmasızdır. Otomatik **bağımlılık taraması** (her yayında çalışan) bu açıkları yakalar ve güncel tutmanı söyler ("tedarik zinciri güvenliği"). **(2) Sırlar** — Seviye 1'de gördüğün gibi sırlar koda girmemeli; ek olarak, hatta bir **sır taraması** ekleyerek yanlışlıkla sızmış bir anahtarı yayına çıkmadan yakalarsın. **(3) Hattın kendisi** — dağıtım hattı güçlü yetkilere sahiptir (üretime erişir); bu yüzden hattın erişimini **en az yetkiyle** sınırla ve hatta erişimi koru, yoksa hat bir saldırı hedefi olur. Ek olarak, sürümleri **imzalamak** yayınlanan kodun gerçekten senden geldiğini doğrular. Güvenlik, yayının ayrı bir adımı değil, hattın her aşamasına gömülü olmalıdır ("kaydır-sola" / shift-left: güvenliği sona bırakma, baştan dahil et). Bu, Modül 7-12'deki güvenlik bilincinin yayın diline çevrilmiş hâlidir.

Yayında ne olur? YAYIN

Yayın güvenliği önlemleri kurulduğunda, hat her yayında kodu ve bağımlılıkları otomatik tarar: bilinen açığı olan bir paket veya yanlışlıkla sızmış bir sır varsa, yayın durur ve sorun düzeltilir. Dağıtım hattının kendisi de yalnızca gereken yetkiyle çalışır, böylece bir hedef hâline gelmez. Sonuç: yalnızca güvenli kod ve güvenilir bağımlılıklar üretime ulaşır; güvenlik, yayının ayrılmaz bir parçası olur.

Alıştırma 12 dk

Yayını güvenli kıl:

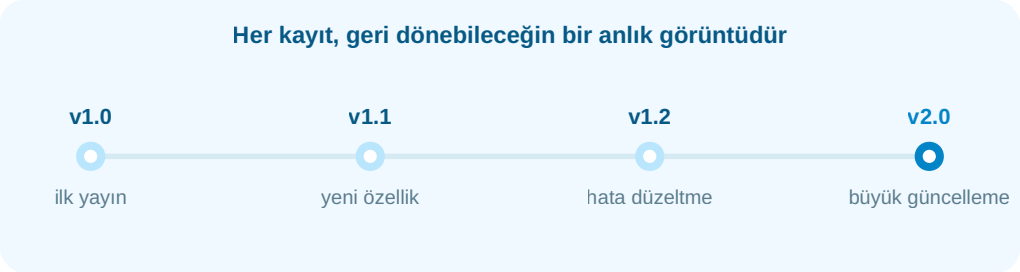
1. Modern saldırıların neden sıklıkla bağımlılıklardan geldiğini açıkla.
2. Sır taramasının ne işe yaradığını yaz.
3. "Güvenliği sona bırakma, baştan dahil et" (shift-left) ne demek, belirt.

BÖLÜM 23

Dokümantasyon ve Sürüm Notları

İyi yazılım, iyi belgelenir. Sürüm notları (changelog) her yayında neyin değiştiğini kaydeder; dokümantasyon ise sistemin nasıl kurulduğunu, çalıştığını ve yayınlandığını anlatır. Bu kayıtlar, geleceğin sana (ve ekibine) bırakılmış değerli bir hediyedir.

Sürümleri belgele



Şema 23.1 — Sürüm geçmişi: her sürüm, neyin değiştiğiyle kayıtlı.

Bir sürüm notu (changelog) örneği

```
## v2.0.0 — 2026-06-01
### Eklendi
- Karanlık tema desteği
### Düzeltildi
- Ödeme sayfasındaki hesaplama hatası
### Değişti (DİKKAT: uyumu bozar)
- Eski API uç noktası kaldırıldı
```

İPUCU

Dokümantasyon, "gelecekteki sana ve ekibine" yazılmış bir mektuptur — bir özelliği neden ve nasıl yaptığını altı ay sonra hatırlamazsın, ama iyi bir kayıt hatırlatır. İki tür özellikle yayınla ilgilidir: **Sürüm notları (changelog)** her yayında **neyin değiştiğini** (eklenen, düzeltilen, değişen — özellikle **uyumu bozan** değişiklikleri) listeler; bu, kullanıcıların ve ekibin bir sürümde ne olduğunu anlamasını sağlar ve anlamsal sürümlemeyle (Seviye 1) doğrudan bağlantılıdır. **Çalıştırma/yayın dokümantasyonu** ise sistemin nasıl kurulduğunu, yapılandırıldığını ve dağıtıldığını anlatır — yeni bir ekip üyesi (veya unutmuş hâlin) bununla hızla devralabilir. İyi dokümantasyonun kuralı: **kodla birlikte güncel tut** (eski/yanlış doküman, hiç dokümandan kötüdür) ve **kısa ve işe yarar** tut (kimse roman okumak istemez). Birçok ekip changelog'u **otomatik** üretir (commit mesajlarından) — bu hem tutarlılık sağlar hem yükü azaltır. Belgelemek "ekstra iş" gibi görünse de, gelecekteki zaman kayıplarını ve hataları önleyerek kendini fazlasıyla öder.

Yayında ne olur? YAYIN

Her sürümde bir changelog tuttuğunda, kullanıcılar ve ekip "bu sürümde ne değişti?" sorusunun yanıtını anında bulur — özellikle uyumu bozan değişiklikler net biçimde işaretlenir. Çalıştırma dokümantasyonuyla, sistemin nasıl yayınlandığı kayıtlı olduğundan, yeni biri (veya aylar sonra sen) süreci tahmin etmek zorunda kalmaz. Sonuç: bilgi kişilerin kafasında değil, erişilebilir kayıtlarda yaşar; ekip daha hızlı ve güvenli çalışır.

Alıştırma 10 dk

Belgele:

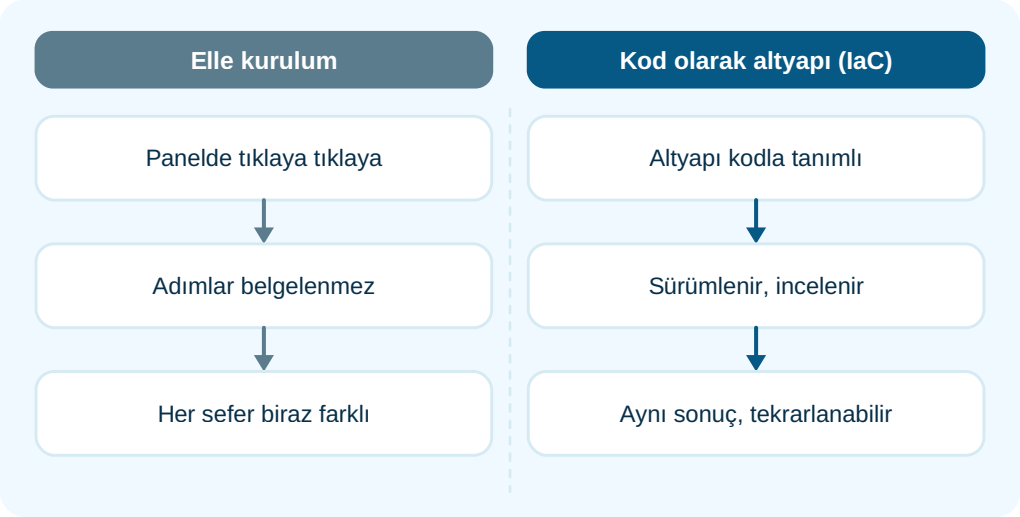
1. Bir changelog'un hangi soruyu yanıtladığını ve neyi mutlaka işaretlemesi gerektiğini yaz.
2. Dokümantasyonu "kodla birlikte güncel tutmak" neden önemli, açıkla.
3. Changelog'u otomatik üretmenin avantajını belirt.

BÖLÜM 24

Otomasyon ve Altyapı (IaC)

Olgun yayın, elle yapılan işleri azaltır. Altyapının kod olarak tanımlanması (Infrastructure as Code / IaC), sunucuları ve ortamları tıklayarak değil, kodla kurmak demektir. Böylece bir ortamı yeniden oluşturmak, bir komut çalıştırmak kadar kolay, tekrarlanabilir ve güvenilir olur.

Elle kurulum vs kod olarak altyapı



Şema 24.1 — IaC: altyapıyı tıklayarak değil, kodla ve tekrarlanabilir kur.

İPUCU

Otomasyonun mantığı şudur: **bir işi ikiden fazla kez elle yapıyorsan, otomatikleştirmeyi düşün**. Yayın dünyasında bunun en güçlü hâli **Altyapının Kod Olması (IaC)**'dır: sunucuları, veritabanlarını, ağ ayarlarını bir panelde elle kurmak yerine, bir **metin dosyasında tanımlarsın** ve bir komutla oluşturursun. Avantajları büyüktür: **tekrarlanabilirlik** (aynı tanım, her yerde aynı ortamı kurar — "staging neden üretimden farklı?" sorunu biter), **sürümlenebilirlik** (altyapı tanımı da Git'te tutulur, değişiklikleri incelenir ve geri alınabilir — tıpkı kod gibi), **belgelenmişlik** (kurulum, kafalarda değil dosyada yazılı), ve **hız** (yeni bir ortamı dakikalar içinde, hatasız kurabilirsin). Elle kurulumun en büyük sorunu "kar tanesi sunucular"dır — her biri zamanla biraz farklılaşan, kimsenin tam nasıl kurulduğunu bilmediği ortamlar; IaC bunu önler. IaC ileri bir konudur ve bu modülde yalnızca tanıştırıyoruz, ama temel fikir tüm seviyelerde geçerlidir: **tekrarlanan, elle yapılan, hataya açık işleri otomatikleştir** — bu, hem güvenilirliği hem hızı artırır ve insanları yaratıcı işe serbest bırakır.

Yayında ne olur? YAYIN

Altyapıyı kod olarak tanımladığında, bir ortamı (sunucu, veritabanı, ağ) kurmak elle tıklamalar yerine bir komut çalıştırmaya dönüşür; aynı tanım her seferinde aynı ortamı üretir. Altyapı değişiklikleri de Git'te sürümlenip incelendiğinden, "kim neyi neden değiştirdi" bellidir ve gerektiğinde geri alınabilir. Sonuç: ortamlar tutarlı, kurulum hızlı ve hatasız, bilgi ise dosyalarda kalıcı olur — elle kurulumun tutarsızlığı ve kırılganlığı ortadan kalkar.

Alıştırma 10 dk

Otomatikleştir:

1. IaC'nin (altyapının kod olması) iki avantajını yaz.
2. "Kar tanesi sunucu" sorununu ve IaC'nin bunu nasıl çözdüğünü açıkla.
3. "Bir işi ikiden fazla kez elle yapıyorsan otomatikleştir" ilkesini kendi cümlenle belirt.

BÖLÜM 25

Ekip İş Akışı ve Kültür

İyi yayın araçları kadar, iyi bir ekip kültürü de önemlidir. Sık, güvenli ve korkusuz yayın yapan ekiplerin ortak özellikleri vardır: küçük adımlar, paylaşılan sorumluluk, suçlamasız öğrenme ve otomasyona güven. Bu kültür, "DevOps" olarak da bilinir.

Sağlıklı yayın kültürü

Şema 25.1 — Sağlıklı yayın (DevOps) kültürünün dört temeli.

İPUCU

Teknik araçlar tek başına yetmez; onları kullanan ekibin **kültürü** belirleyicidir. Sağlıklı yayın kültürünün temelleri: **Küçük ve sık yayın** — büyük, seyrek ve korkulu yayınlar yerine, küçük ve sürekli yayınlar (her biri az risk taşır, sorun çıkarsa kolay bulunur). **Paylaşılan sorumluluk** — "geliştiriciler kod yazar, başkaları yayınlar/işletir" ayrımı yerine, ekip kodu hem yazar hem yayınlar hem sahiplenir ("DevOps"un özü budur: geliştirme ve işletimi birleştirmek). **Suçlamasız öğrenme** — hatalar (Seviye 4'teki olaylar) ceza değil, öğrenme fırsatı; insanlar hatalarını gizlemek yerine açıkça paylaşıp süreci iyileştirir. **Otomasyona güven** — tekrarlanan işleri (test, dağıtım, altyapı) makinelere bırakıp, insanları yaratıcı ve değerli işe ayırmak. **Psikolojik güvenlik** — insanların "bu çalışmıyor", "bir hata yaptım", "bunu anlamıyorum" diyebildiği ortam, en hızlı öğrenen ve en güvenilir yayın yapan ortamdır. Bu kültür bir gecede kurulmaz; küçük adımlarla, güven inşa ederek büyür. Ama sonunda, hız ve güvenilirliği birlikte mümkün kılan şey araçlar değil, **bu kültürdür**.

Yayında ne olur? YAYIN

Sağlıklı bir yayın kültüründe ekip küçük ve sık yayınlar yapar (her biri düşük riskli), kodu birlikte sahiplenir, hatalardan suçlama olmadan ders çıkarır ve tekrarlayan işleri otomasyona bırakır. Bu ortamda insanlar açıkça konuşur, sorunları erken paylaşır ve sürekli iyileşir. Sonuç: yayın korkulan bir olay olmaktan çıkar; ekip hızlı, güvenli ve sürdürülebilir biçimde değer üretir.

Alıştırma

10 dk

Kültürü kur:

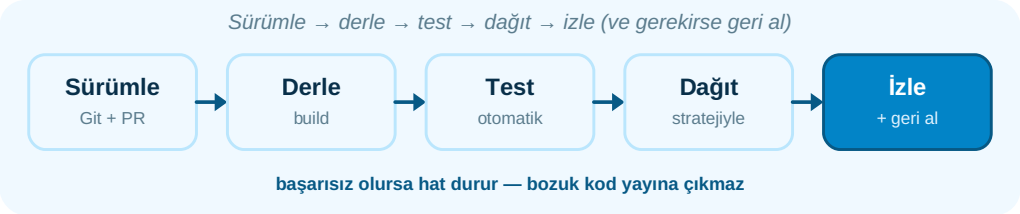
1. Sağlıklı yayın kültürünün dört temelini yaz.
2. "DevOps"un (geliştirme + işletimi birleştirmek) ne demek olduğunu açıkla.
3. "Psikolojik güvenliğin" iyi yayınla ilişkisini kendi cümlenle belirt.

BÖLÜM 26

Bitirme: Yayın Hattı Kontrol Listesi

Tüm modülü, sağlam bir yayın süreci kurmak için kullanacağın bir kontrol listesinde topluyoruz. Bu liste, "çalışan kod"u "güvenle, sürekli ve geri alınabilir biçimde yayınlanan kod"a dönüştüren adımların özetidir.

Sağlam yayın hattı



Şema 26.1 — Tam yayın hattı: kodu güvenle ve sürekli dünyaya açar.

Yayın hattı kontrol listesi

```
[ ] Tüm kod sürüm kontrolünde (Git), PR ile inceleniyor
[ ] Sürümler anlamsal olarak numaralanıyor (semver)
[ ] Sırlar koddan ayrı (ortam değişkeni), depoda yok
[ ] Otomatik testler her commit'te çalışıyor (CI)
[ ] Dağıtım otomatik ve tutarlı (CD)
[ ] Ortamlar ayrı: geliştirme / staging / üretim
[ ] Hızlı, denenmiş geri alma yolu var
[ ] İzleme, loglama ve hata takibi aktif
[ ] Bağımlılık + sır taraması hatta gömülü
[ ] Sürüm notları (changelog) tutuluyor
```

İPUCU

Bu kontrol listesi, modülün tamamının özüdür; her proje için yayın sürecini kurarken başvur. Bu modülle birlikte **kodu profesyonelce dünyaya açmayı** öğrendin: sürüm kontrolü ve dal/PR akışı, sürümleme, build, ortamlar ve sırlar (Seviye 1); CI/CD ile otomasyon (Seviye 2); geri alma, özellik bayrakları, izleme, loglama, hata takibi (Seviye 3); olay yönetimi, yayın güvenliği, dokümantasyon, IaC ve kültür (Seviye 4). Bütünsel kavrayış şudur: **iyi yayın, tek bir an değil, bir sistemdir** — kodun güvenle, sık sık ve geri alınabilir biçimde kullanıcıya ulaşmasını sağlayan, otomasyon ve disiplinden örülmüş bir bütün. En önemli ilke baştan beri aynı: yayını **nadir ve korkulu** bir olaydan, **sık, sıradan ve güvenli** bir rutine dönüştürmek. Çünkü hızlı ve güvenle yayınlayan ekipler, daha hızlı öğrenir, daha çabuk düzeltir ve kullanıcılarına daha hızlı değer ulaştırır. Bir sonraki modülde (Kodlamada Yapay Zekâ), bu güçlü temelin üstüne, yazılım üretimini hızlandıran yapay zekâ araçlarını sorumlu biçimde kullanmayı ekleyeceğiz.

Yayında ne olur? YAYIN

Kontrol listesini baştan sona uyguladığında, projen yalnızca "çalışan kod" değil, güvenle yayınlanan bir sistem olur: her değişiklik kayıtlı ve incelenmiş, her yayın otomatik test edilmiş, dağıtım tutarlı, geri dönüş yolu hazır ve yayın sonrası her şey izleniyor. Bir sorun çıktığında hızla görülür ve geri alınır; her yayın bir öncekinden biraz daha güvenli olur. Sonuç: bir fikri, dünyanın güvenle erişebileceği ve sürekli iyileşen bir ürüne dönüştürmenin tüm zincirine sahip olursun.

Alıştırma 20 dk

Kendi hattını kur:

1. Hayalî bir proje için yayın hattı kontrol listesini doldur.
2. Her maddenin neden önemli olduğunu bir cümleyle yaz.
3. Sürümle → test → dağıt → izle → geri al zincirini kendi cümlelerinle özetle.
4. Bu modülde öğrendiğin en değerli üç şeyi belirt.

EK

Yayınlama Terimleri Sözlüğü

En sık kullanılan yayınlama, sürüm yönetimi ve dağıtım terimleri. Bir başvuru kaynağı olarak saklayabilirsin.

Yayınlama (deploy)	`Kodu kullanıcıya açma`	Sürüm kontrolü	`Git: kayıt + geri alma`
Commit	`Geri dönülebilir kayıt`	Dal (branch)	`Paralel çalışma alanı`
Pull request (PR)	`İncele + birleştir isteği`	Semver	`BÜYÜK.KÜÇÜK.YAMA`
Build	`Yayına uygun çıktı üret`	Ortam	`Geliştirme/Staging/Üretim`
Ortam değişkeni	`Koddan ayrı ayar/sır`	CI	`Sürekli entegrasyon`
CD	`Sürekli dağıtım`	Pipeline	`Otomatik yayın hattı`
Rollback	`Önceki sürüme dönme`	İzleme	`Yayını sürekli gözleme`

Yayın zinciri

YAYIN

Profesyonel yayın bir zincirdir: **sürüm kontrolü** (Git) ile her değişiklik kaydedilir, **dallar ve PR** ile incelenip birleştirilir, **sürüm numarasıyla** etiketlenir, **yapı (build)** ile yayına uygun çıktıya dönüşür, **ortamlardan** (geliştirme → staging → üretim) güvenle terfi eder ve **ayarlar/sırlar** koddan ayrı tutulur. İleri seviyelerde bu zinciri **otomatikleştiren** CI/CD hatlarını, **güvenli** dağıtım stratejilerini ve geri almayı, yayını sağlıklı tutan **izleme/loglama**yı ve olay yönetimini ele alacağız. Hepsinin ortak amacı tek: kodu **sık, güvenli ve geri alınabilir** biçimde dünyaya açmak.