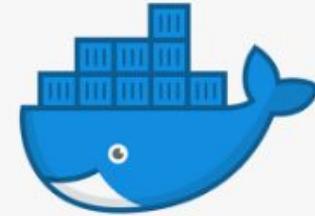
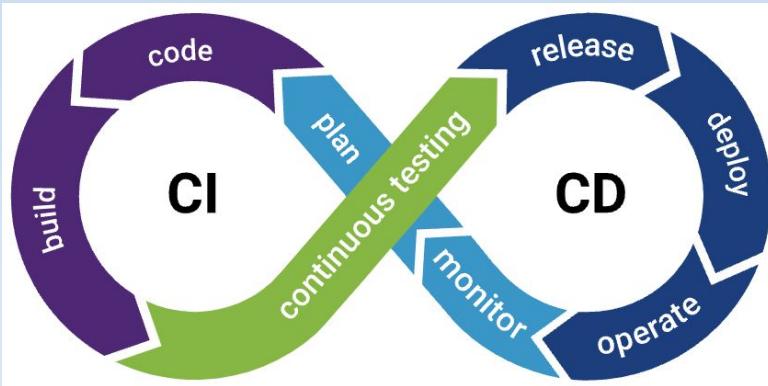




# DevOps Fundamentals

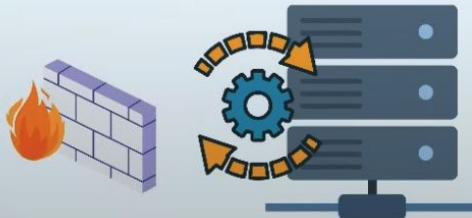
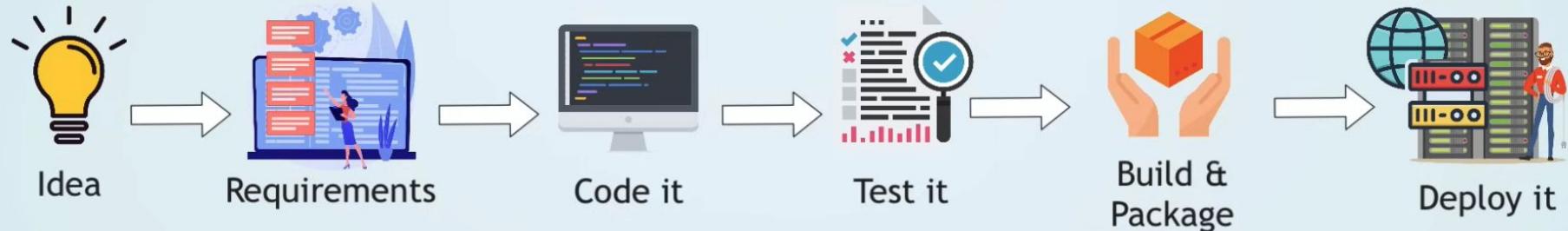


Hakan BAYRAKTAR

# DevOps Temelleri

- DevOps Nedir?
- DevOps'un Faydaları
- DevOps Mühendisi Kimdir?
- DevOps Araçları
- DevOps Yaşam Döngüsü
- Docker
- Kubernetes
- Git ve GitHub
- CI/CD Entegrasyonu (Jenkins, GitHub Action, GitLab CI)
- İzleme ve Günlükleme (Grafana, Prometheus, ELK Stack)

# Typical Software Release Process



Configure the server

- ▶ Installing tools
- ▶ Deploy application
- ▶ Configure Firewall



Any problems with the application?

Are users experiencing any issues?

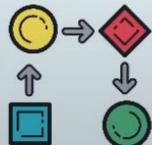
Can the application handle high user load?



# Software versioning



## Improvements



- ▶ Add new features!
- ▶ Fix bugs
- ▶ Optimize performance



# Software versioning



# Cloud Services

## Cloud Services

### Traditional On-Premise (On-Prem)

- Applications
- Data
- Runtime
- Middleware
- Operating System
- Virtualization
- Servers
- Storage
- Networking

### Infrastructure as a Service (IaaS)

- Applications
- Data
- Runtime
- Middleware
- Operating System
- Virtualization
- Servers
- Storage
- Networking

### Platform as a Service (PaaS)

- Applications
- Data
- Runtime
- Middleware
- Operating System
- Virtualization
- Servers
- Storage
- Networking

### Software as a Service (SaaS)

- Applications
- Data
- Runtime
- Middleware
- Operating System
- Virtualization
- Servers
- Storage
- Networking

■ Self Managed

■ Managed By Vendor

# Yazılım Teslim Modeli Nasıl Değişti?

Old software delivery model

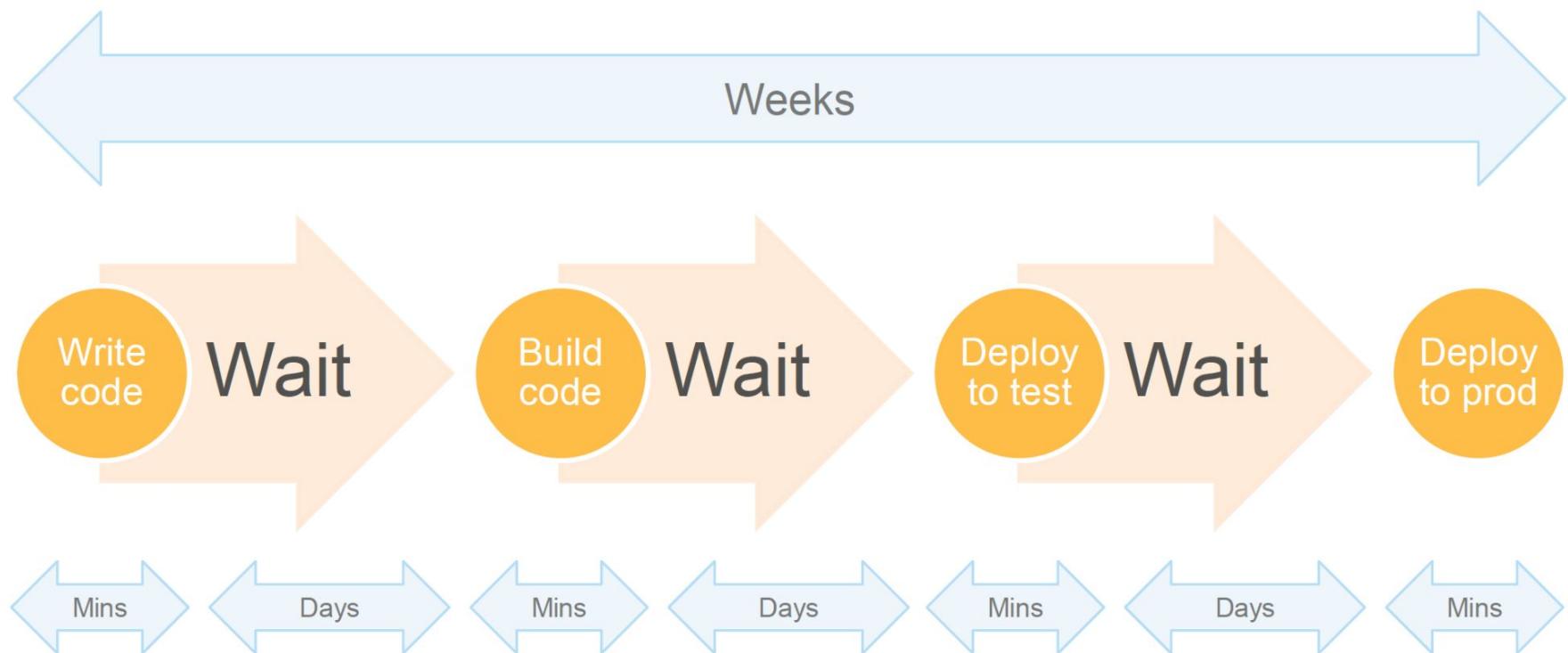


New software delivery model



# Yazılım teslim modeli büyük ölçüde değişti

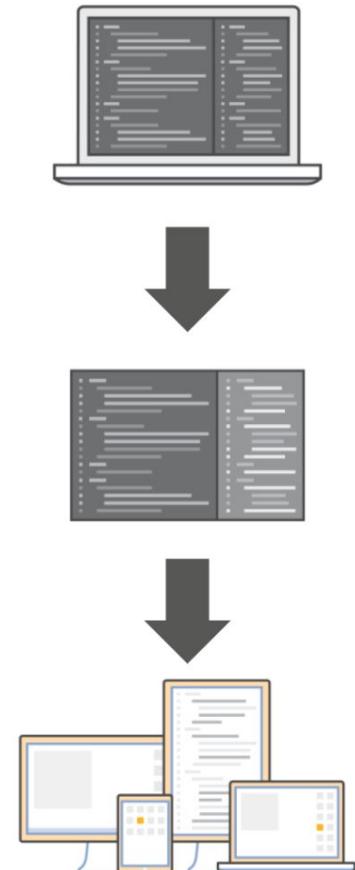
We were just waiting



# Software moves faster today

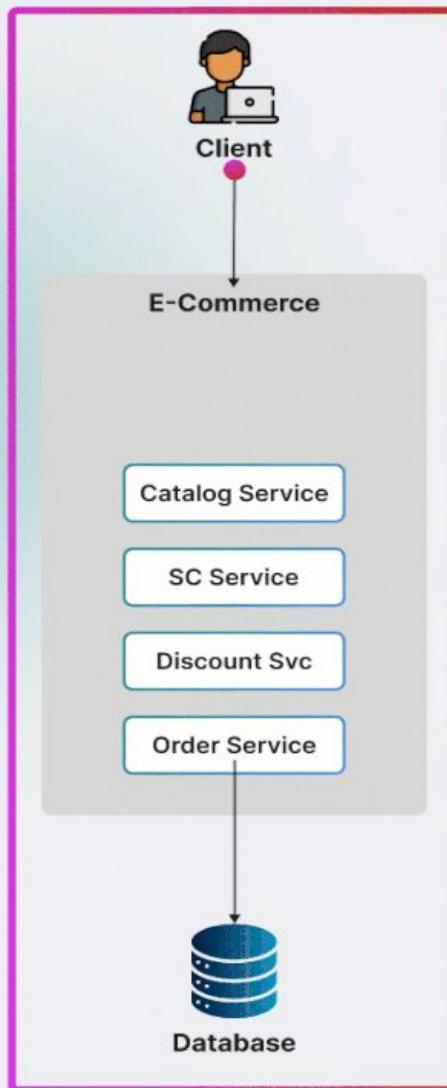
Software creation and distribution is easier and faster than ever

- Startups can now take on giants with little to no funding ahead of time
- Getting your software into the hands of millions is a download away
- Your ability to move fast is paramount to your ability to fight off disruption

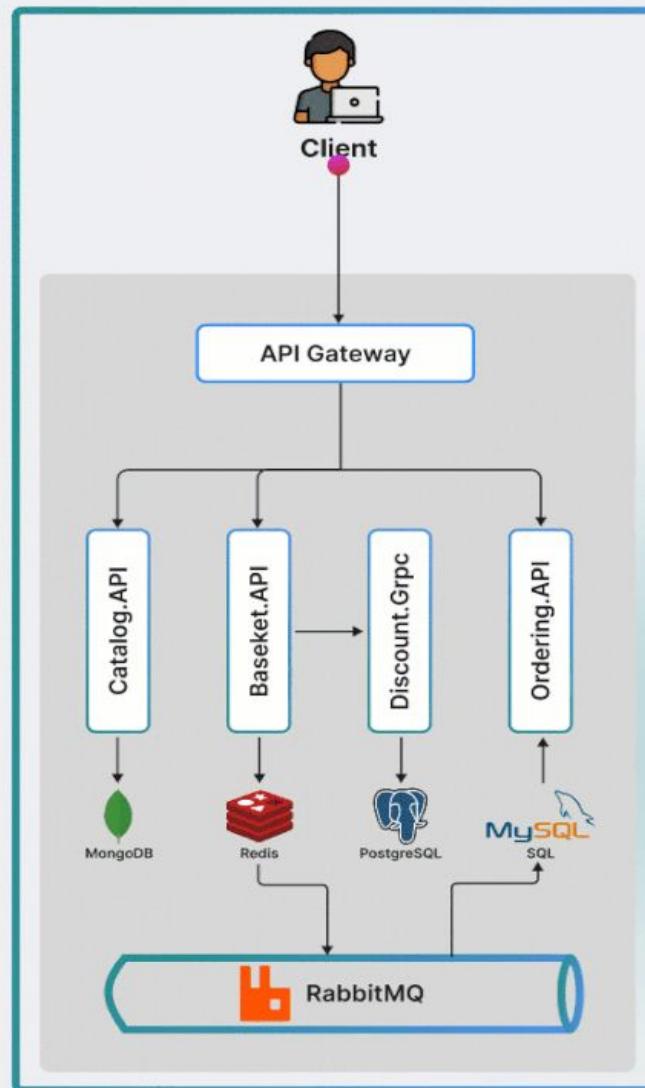


# MONOLITHIC VS MICROSERVICES ARCHITECTURE

## MONOLITHIC

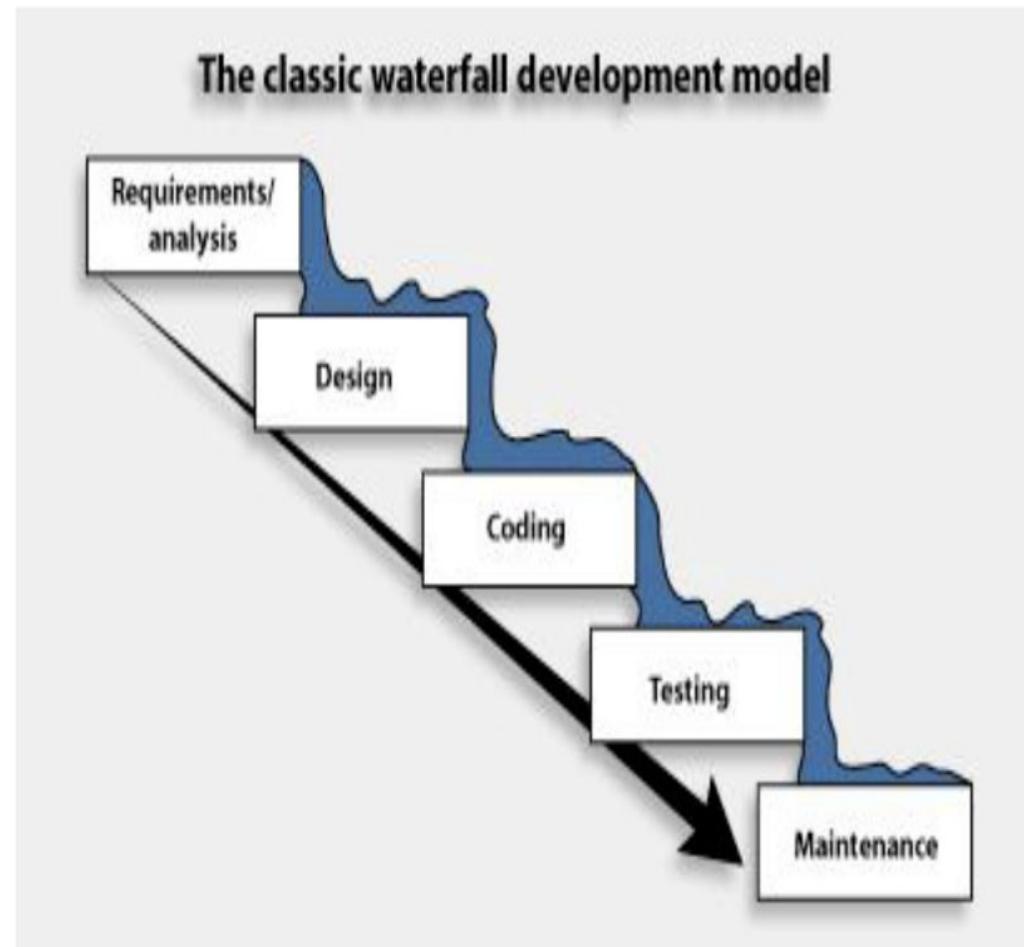


## MICROSERVICES



# Klasik Yazılım Geliştirme Süreci

- 1-Gereksinimlerin Belirlenmesi
- 2-Tasarımın Tamamlanması
- 3-Kodlama ve Birim Testleri
- 4-Fonksiyonel ve Entegrasyon Testleri
- 5-Kabul Testlerinin Yapılması ve Dağıtım
- 6-Bakım



# Agile Yazılım Geliştirme Süreci

- Her İterasyon Kapsam ve Hedefleri Belirler
- Sürekli Tasarım ve Geliştirme
- Sürekli Test Edilme
- Sık Sık Yazılım Yayınlama
- Geri Bildirim ve Performans Ölçümleriyle İyileştirme

# Agile Nedir?

Agile, yazılım geliştirme süreçlerini daha esnek, hızlı ve etkili hale getirmek için tasarlanmış bir metodolojidir.

Agile, **değişime hızlı yanıt verme, müşteri memnuniyetini artırma ve sürekli iyileştirme** felsefesine dayanan bir yöntemdir. Büyük bir projenin tamamını önceden planlamak yerine, küçük parçalar halinde, iteratif ve artımlı bir şekilde çalışmayı teşvik eder.

# Agile Nedir Değildir?

## Nedir?

- **Müşteri Odaklılık:** Müşteri ihtiyaçlarına göre esnek bir şekilde yön değiştirebilir.
- **İşbirliği:** Takım üyeleri arasında sürekli iletişim ve işbirliğini teşvik eder.
- **Hızlı Teslimat:** Proje ilerledikçe sürekli teslimat yaparak erken geri bildirim almayı sağlar.

## Değildir:

- **Sadece Hızlı Çalışmak:** Agile, hızlı çalışmayı değil, etkili ve kaliteli bir şekilde hızlı çalışmayı hedefler.
- **Her Şeyin Tamamını Planlamak:** Tüm projeyi baştan sona planlamak yerine, adım adım ilerler ve gerekiğinde değişiklikler yapar.
- **Sadece Yazılım Geliştirme:** Agile, yalnızca yazılım projeleri için değil, birçok sektörde ve alanda uygulanabilir.

# Agile Ana Bileşenler

**Takım:** Çapraz fonksiyonel ekipler, yani yazılımcılar, tasarımcılar ve diğer paydaşlar bir arada çalışır.

**Iterasyon:** Projeler kısa döngüler (sprint) içinde tamamlanır. Her sprint sonunda bir ürün geliştirilir.

**Geri Bildirim:** Her iterasyon sonrası kullanıcı ve paydaşlardan geri bildirim alınarak iyileştirmeler yapılır.

**Planlama ve Değerlendirme:** Her sprint öncesi planlama, sonrası değerlendirme yapılır.

## Tablo 1: Temel Agile Kavramları

Kavram	Açıklama
Sprint	Kısa ve belirli bir süre içinde (genellikle 1-4 hafta) gerçekleştirilen geliştirme döngüsü.
Backlog	Projede yapılacak işlerin ve özelliklerin listesidir. İki türü vardır: Product Backlog (ürün) ve Sprint Backlog (sprint).
Scrum	Agile yöntemlerinden biri olup, takımın sürekli olarak çalışmasını sağlayan bir çerçeve.
User Story	Kullanıcı gereksinimlerini basit bir dilde tanımlayan kısa açıklamalardır.
Daily Standup	Ekip üyelerinin her gün bir araya gelerek güncellemeleri paylaştığı kısa toplantıdır.
Retrospektif	Sprint sonrasında ekip üyelerinin süreçlerini değerlendirdiği toplantıdır.
Velocity	Ekiplerin bir sprintte tamamladıkları iş miktarını ölçen bir metriktir.

## Agile Manifesto'nun 12 Prensibi

- Müşteri Memnuniyeti:** Müşterinin memnuniyeti, erken ve sürekli bir şekilde değerli yazılım sunmakla sağlanır.
- Değişime Açıklık:** Değişen gereksinimlere, geliştirme sürecinin her aşamasında yanıt verilmelidir, hatta proje sonrasında bile.
- Sık Teslimat:** Çalışan yazılım, sık bir şekilde (haftalar, aylar) sunulmalıdır. Mümkünse en kısa süre içinde.
- İşbirliği:** İşverenler ve geliştiriciler, projede çalışırken günlük işbirliği yapmalıdır.
- Motivasyonlu Ekipler:** En iyi sonuçlar, kendine güvenen bireylerden oluşan motivasyonu yüksek ekiplerden elde edilir.
- Yüz Yüze İletişim:** Proje ekibinin tüm üyeleri arasında en etkili iletişim yüz yüze iletişimdir.
- Çalışan Yazılım:** Projenin temel ölçütü, çalışan yazılımdır.
- Sürdürülebilir Geliştirme:** Ekipler, sürdürülebilir bir geliştirme hızını desteklemelidir. İşverenler, geliştiriciler ve kullanıcılar, sürekli bir tempoda çalışmalıdır.
- Teknik Mükemmellik:** Teknik mükemmeliyet ve iyi tasarım, artırılmış esnekliğe yol açar.
- Basitlik:** Yapılacak işlerin en azını yaparak, işi mümkün olduğunda basit tutmak en iyisidir.
- Kendini Organize Eden Ekipler:** En iyi mimariler, gereksinimler ve tasarımlar, kendini organize eden ekipler tarafından ortaya çıkar.
- Düzenli Geliştirme:** Takımlar, belirli aralıklarla (örneğin, her sprint sonunda) çalışma süreçlerini gözden geçirmeli ve kendilerini geliştirmeye için ayarlamalar yapmalıdır.

# What is Devops?

DevOps is ...

**DevOps: A culture shift or movement that encourages great collaboration (aka teamwork) to foster building better quality software more quickly with more reliability.**

DevOps is NOT:

- A role, person, or organization
- Something only systems administrators do
- Something only developers do
- Writing Chef and Puppet scripts
- Tools

DevOps is a way of building software while still getting a speed to market without sacrificing quality and reliability.

# What is Devops?



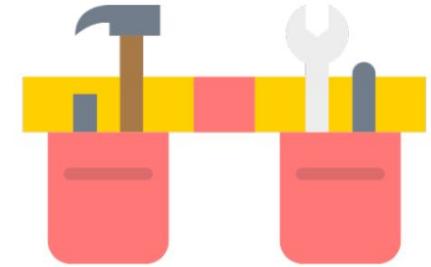
Way of software development



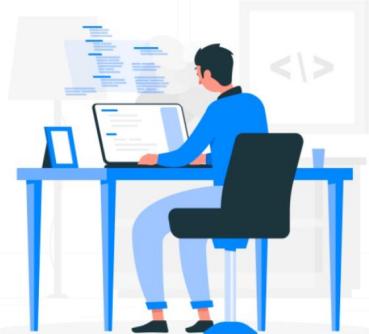
Values & Principle



Methodologies



Tools



Developers

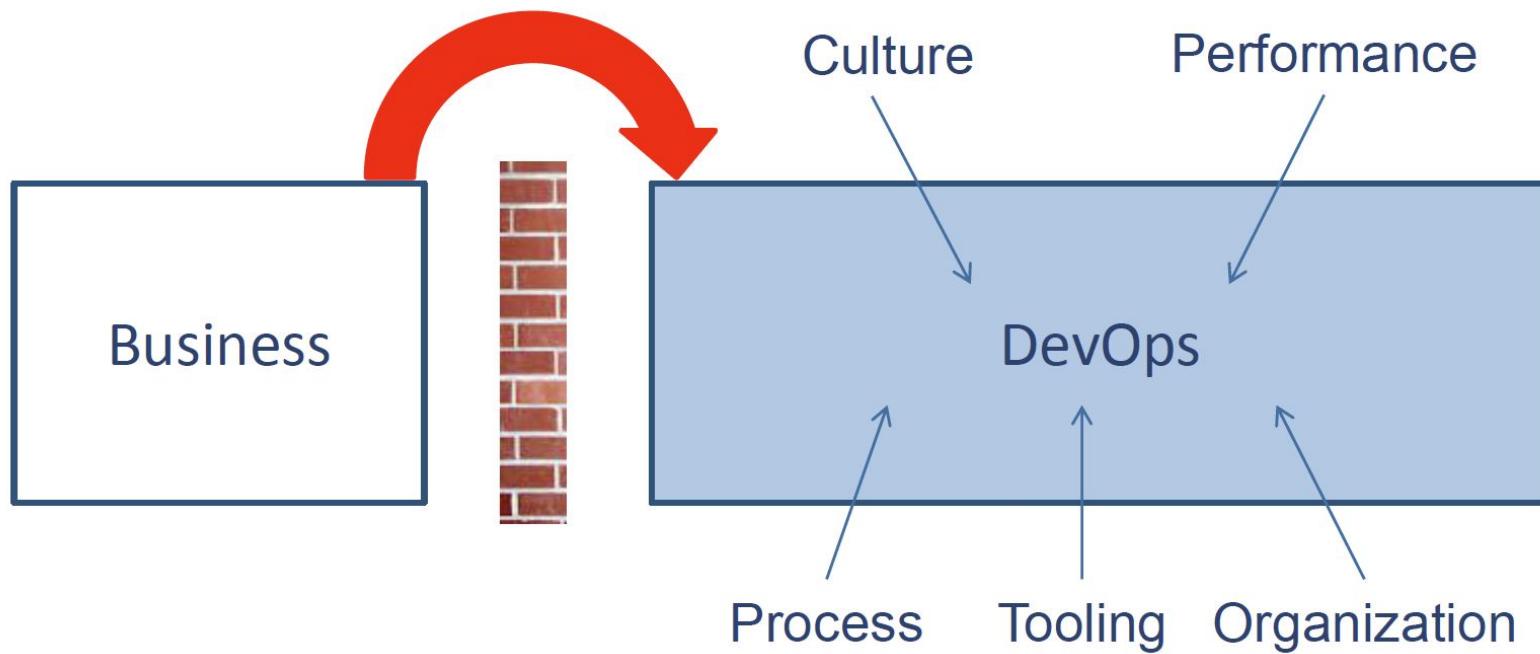


Operations

# Devops'un Ana Hedefleri

DevOps, yazılım geliştirme ve operasyon ekiplerini bir araya getirerek dört temel hedefe ulaşmayı amaçlayan bir hareket veya yaklaşımdır:

- Improve **Quality** of delivery (First time right)
- Deliver higher **Business Value** (Customer)
- Increase **Speed** of delivery (Performance)
- Improve **Efficiency** (Reduce Cost)



# DevOps is the solution to a problem

Automation

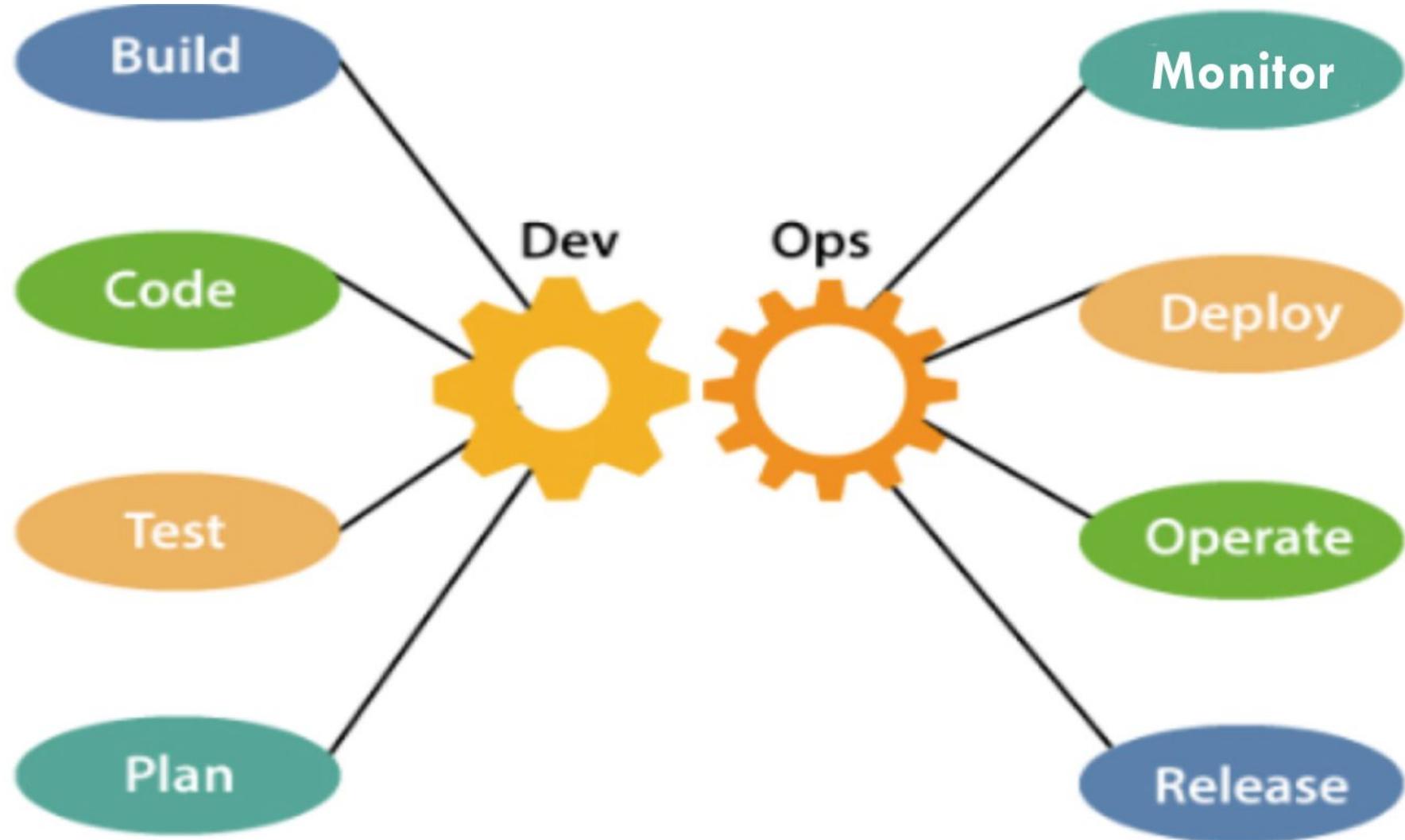
Performance

Culture

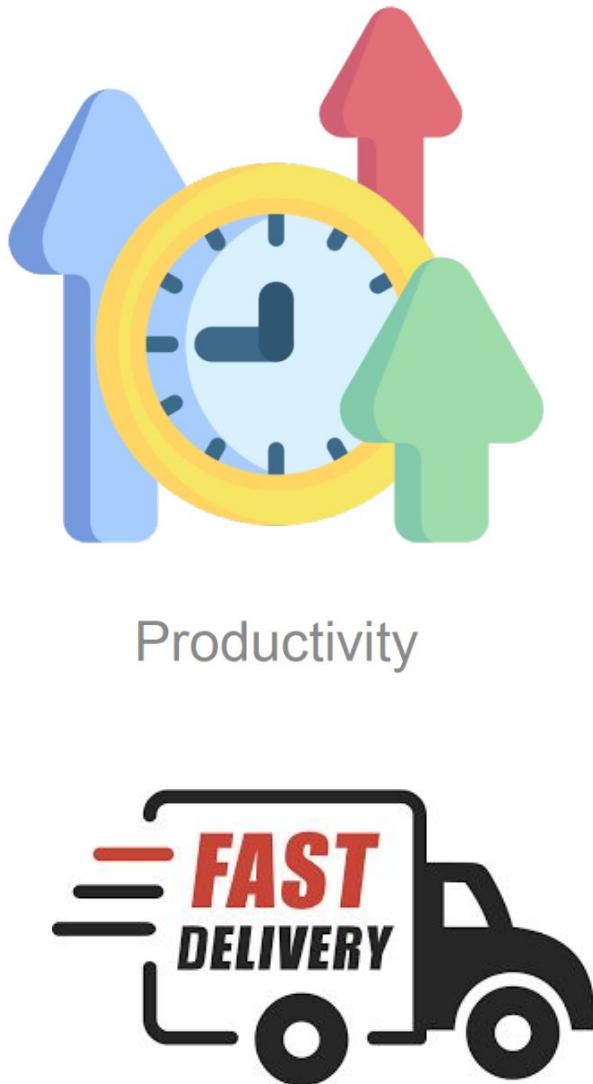
Flow

Organisation

# DevOps Components



# DevOps'un Sağladığı Faydalar



Productivity



Security



Excellence



Improved collaboration



Returns



Reliability



Speed

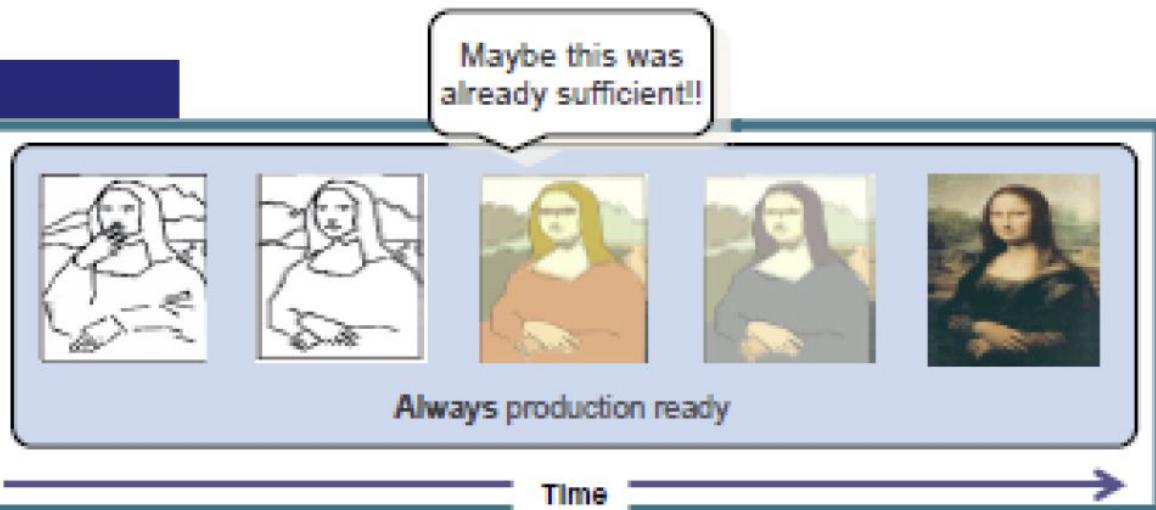


# DevOps Phases

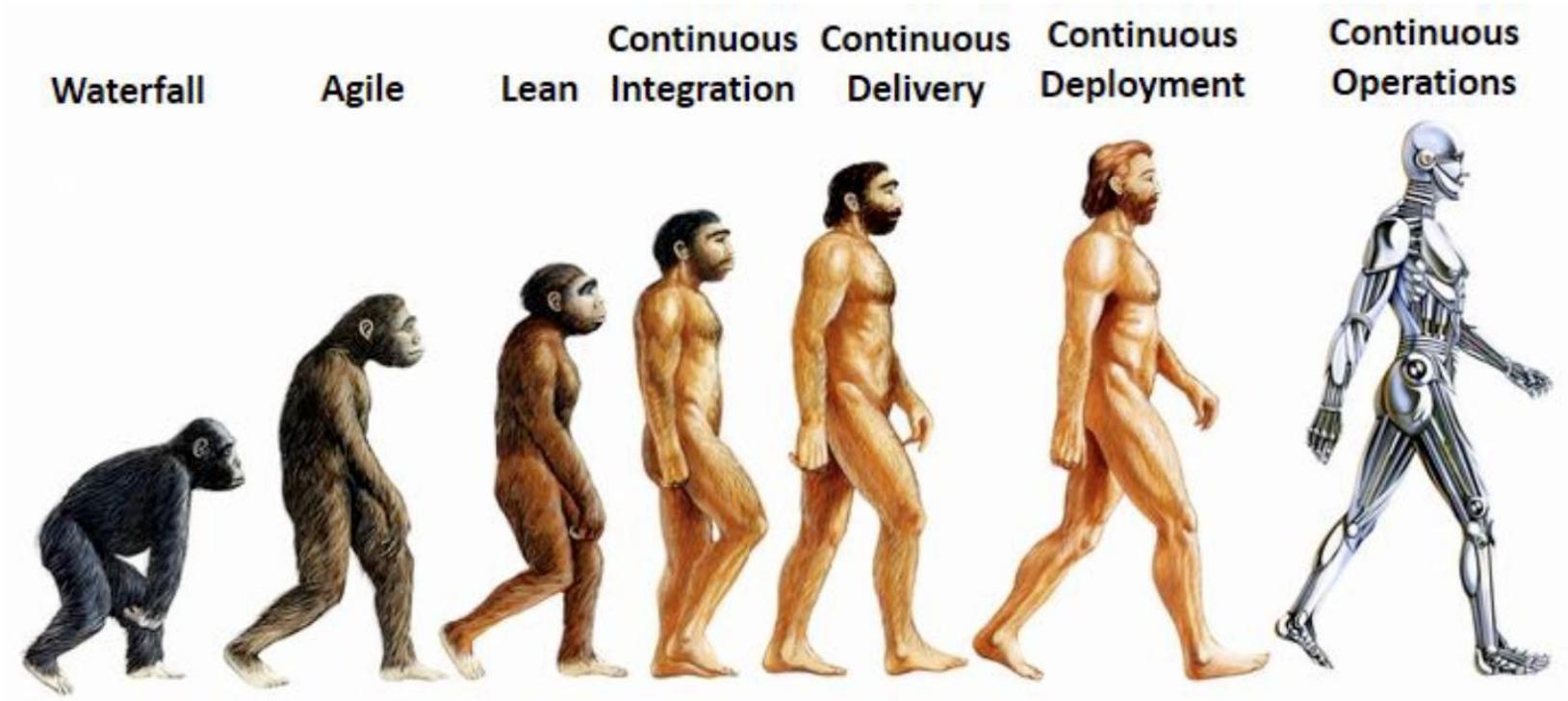
## TRADITIONAL



## AGILE



# DevOps is an evolution



# DevOps is an evolution

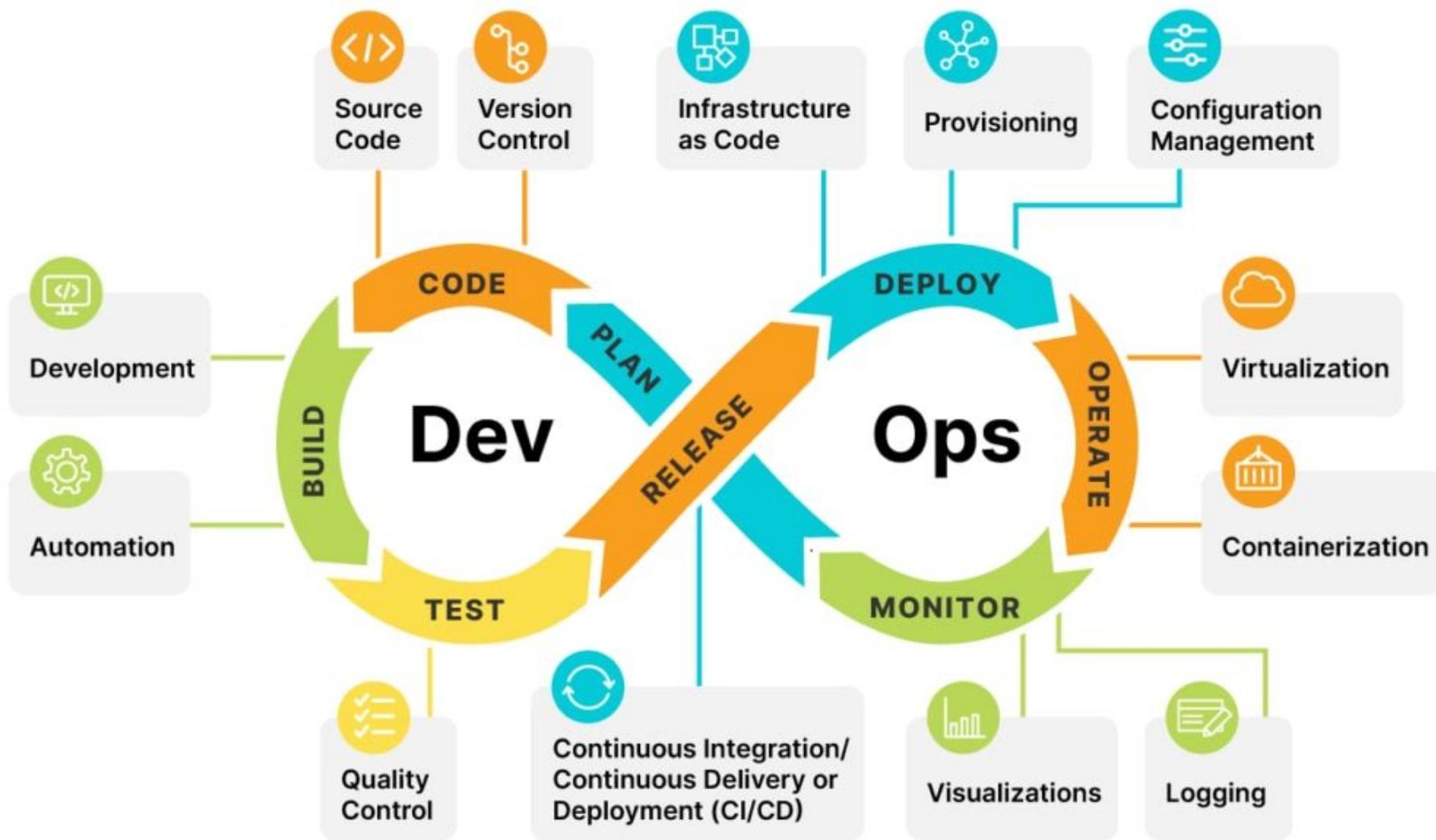
## New Way of Delivering Software

Old Way	New Way
Focus on Ship Date	Focus on Working Software
Prolong Deployments	Release More Frequently
Heroic Efforts	Repeatable & Predictable Processes
Done when code is built	Never done – Always On
Ops runs applications	Everyone is responsible
Ops and Security dictate	Ops and Security Participate
Story Points	Customer Satisfaction
Handoffs	Feedback Loops

# Yazılım teslimatındaki eski ve yeni yaklaşımalar

Özellik	Eski Yaklaşım	Yeni Yaklaşım
Geliştirme Süreci	Waterfall	Agile veya DevOps
Teslimat Sıklığı	Uzun döngüler, genellikle yıllık	Kısa döngüler, genellikle haftalık veya aylık
Geri Bildirim	Geç aşamada, ürün tamamlandıktan sonra	Sürekli, her sprint veya iterasyon sonunda
Belge	Detaylı önceden hazırlanmış belgeler	Daha az belge, işleyen kod ve iletişim
Test Süreci	Son aşamada, genellikle ürün bitiminden sonra	Sürekli entegrasyon ve sürekli test
Değişiklik Yönetimi	Zor, değişiklikler pahalı ve karmaşık	Esnek, değişiklikler kolayca uygulanabilir
Müşteri Katılımı	Az, proje başında ve sonunda	Sürekli, her iterasyonda müşteri geri bildirimi
Risk Yönetimi	Geç tespit edilir, büyük riskler son aşamada	Erken ve sürekli tespit edilir, küçük riskler

# Devops Lifecycle



# DEVOPS METRICS

Mahesh M

## Plan

1. Sprint Burndown
2. Team Velocity
3. Sprint Goal Success
4. Cycle time
5. Work in Progress
6. Epic Burndown
7. Lead time
8. Backlog Estimation
9. Release Burndown
10. Portfolio Planning

## Code

1. Code Review
2. Code Churn
3. Code Quality
4. Technical Debt
5. Code Contribution
6. Lines of Source code
7. Maintainability Index
8. Number of PRs
9. Code Commits
10. Code Coverage

## Build

1. Build Success Rate
2. Build Duration
3. Failed Builds
4. Build Broken Time
5. Build over time
6. Pull requests
7. Build Frequency
8. Build History
9. Pipeline Monitor
10. Queue Time

## Test

1. Test Coverage
2. Tests Pass Vs Failed
3. Defect metrics
4. Test Duration
5. Defect Escape Ratio
6. Vulnerability Report
7. Defects severity
8. Test Flakiness
9. Test build over time
10. Defect Distribution



## Release

1. Release Duration
2. Number of releases
3. Successful releases
4. Features packaged
5. Release Frequency
6. Success Ratio
7. Release Stability
8. Defect Escaped
9. Release Backout
10. Outages

## Deploy

1. Deployment Frequency
2. Time to Deploy
3. Rollback Frequency
4. Deployment Lead Time
5. Number of incidents
6. MTTR
7. Cost / Release
8. Failed Deployment
9. Production downtime
10. Change Failure Rate

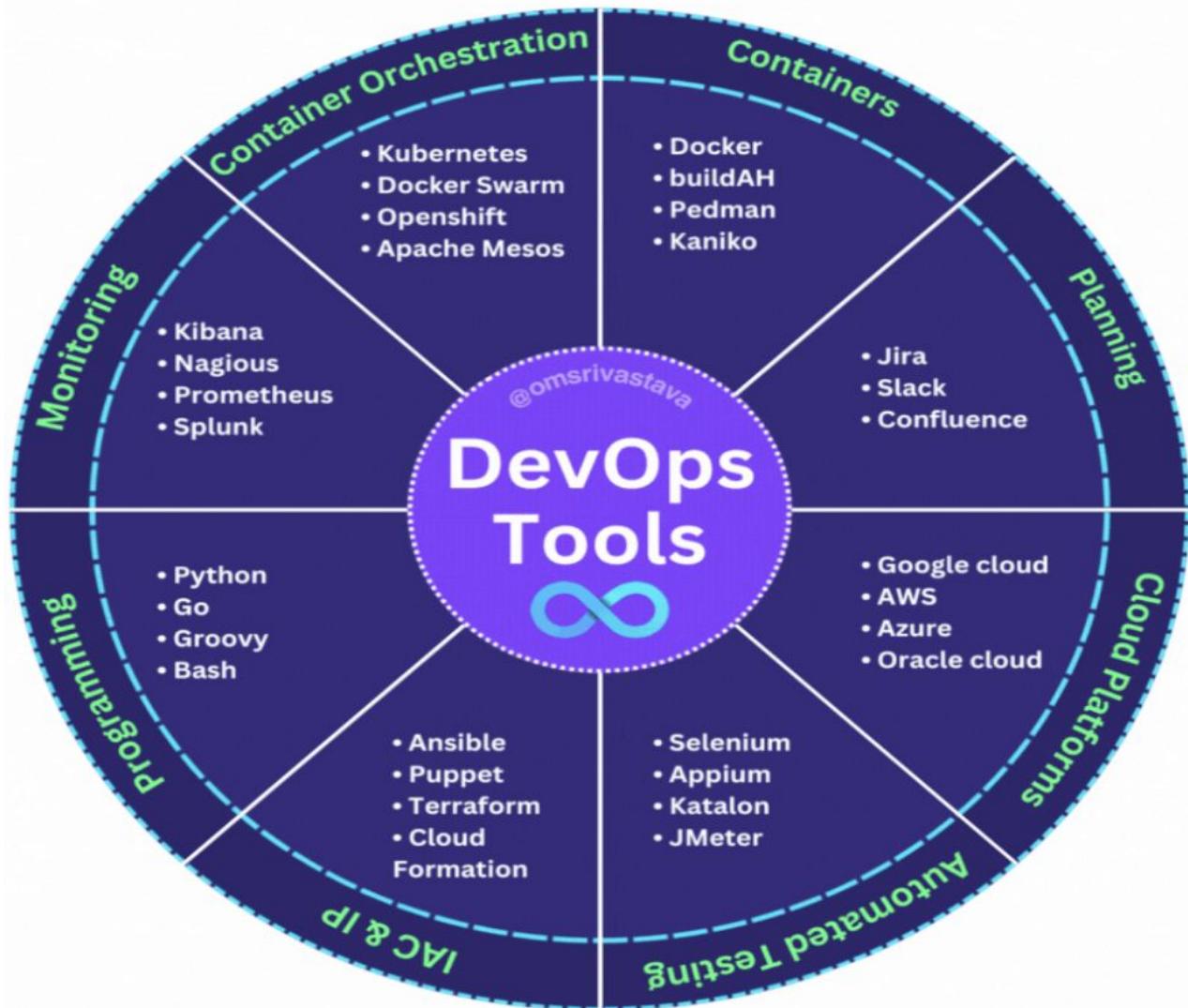
## Operate

1. Customer feedback
2. Customer tickets
3. Configuration failures
4. Downtime
5. Uptime metrics
6. Usage & Traffic
7. Error Budget
8. Performance Score
9. Cost - Product Feature
10. Retention Rate

## Monitor

1. System Uptime
2. SLA, SLI, SLO
3. Performance
4. Mean Time - Failures
5. Mean Time to Detect
6. Mean Time to Repair
7. Error Rate
8. Latency
9. Security Incident
10. Infrastructure Cost

# DevOps Tools



# Who is Best?



Etsy

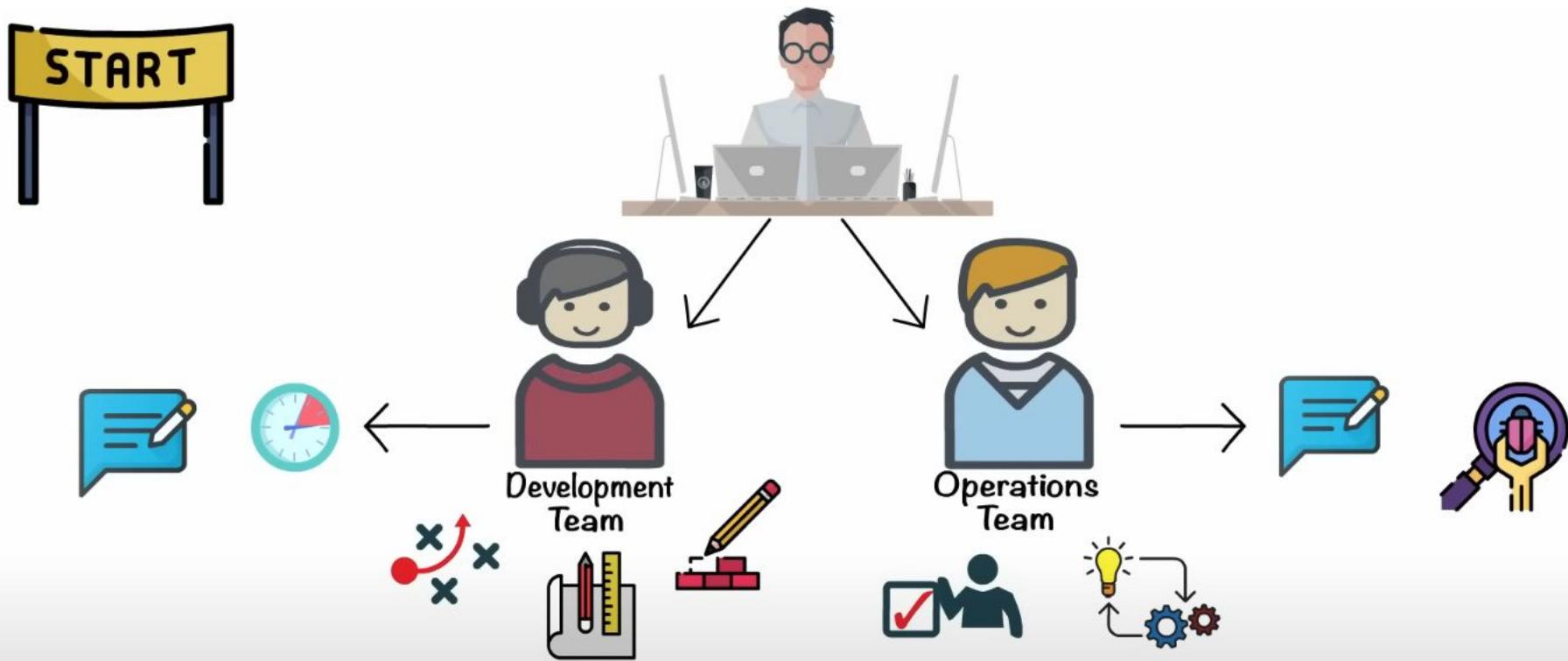


Spotify®

"Amazon has become famous for its high-velocity deployment of more than 2500 software releases per day across its various Cloud solution offerings. And there are similar storied successes at eBay, Etsy, Facebook, Netflix, Spotify, Twitter."  
<http://blog.saugatucktechnology.com/>

# Who is a DevOps Engineer?

DevOps mühendisleri, hem IT operasyonları hem de yazılım geliştirme alanlarında bilgi sahibi olan profesyonellerdir.



# Who is a DevOps Engineer?

- **Güçlü iletişim becerilerine sahip olmaları** sayesinde farklı ekiplerle etkili bir şekilde çalışabilirler.
- DevOps mühendisleri, kodlama, dağıtım, test, bakım ve güncellemeler gibi süreçlerin **her aşamasında aktif rol alırlar**.
- **Farklı ekip rollerini anılarlar** ve bu rollerin gereksinimlerine uygun araçlar ve uygulamalar geliştirirler.
- **DevOps metodolojilerini** uygulayarak iş akışlarını optimize eder ve verimliliği artırırlar. Sürekli entegrasyon ve sürekli teslimat gibi yöntemlerle yazılım geliştirme süreçlerini daha akıcı hale getirirler.



# Devops Mühendisinin Görevleri

- **Otomasyon:** Tekrarlayan işlemleri otomatikleştirerek verimlilik artırmak.
- **CI/CD Pipeline Kurma:** Sürekli entegrasyon ve dağıtım süreçlerini yönetmek.
- **Konteyner Teknolojileri:** Docker ve Kubernetes kullanarak uygulamaları yönetmek.
- **IaC:** Altyapıyı kodla yönetmek
- **Monitoring ve Log Yönetimi:** Sürekli izleme araçları kullanarak sistemlerin sağlığını kontrol etmek.
- **Cloud Yönetimi:** AWS, Azure gibi bulut platformlarını etkin bir şekilde kullanmak.



# DevOps Mühendisinin Uzmanlaşması Gereken Konular

- **CI/CD Araçları:** Jenkins, GitLab CI/CD
- **Konteyner yönetimi** Kubernetes ve Docker
- **IaC:** Terraform, Ansible
- **Monitoring Araçları:** Prometheus, Grafana
- **Cloud Platformları:** AWS, GCP, Azure



# DevOps Mühendisi İçin Sertifikalar

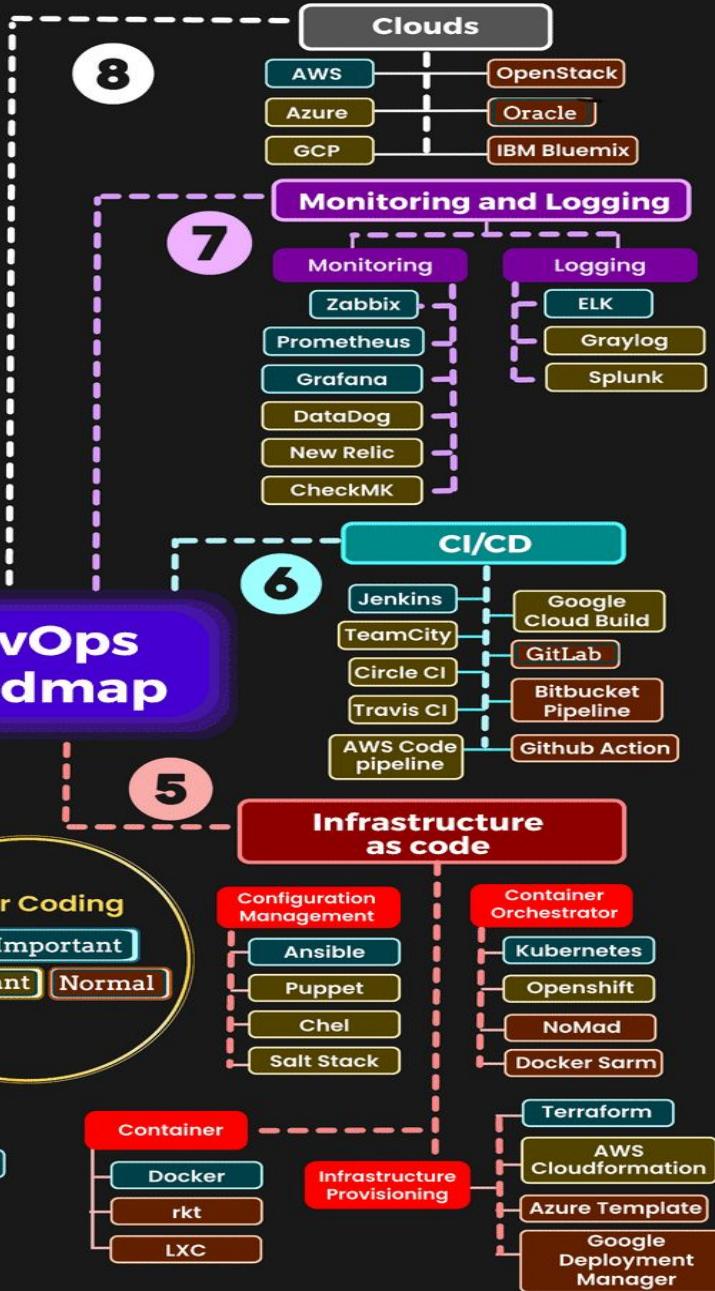
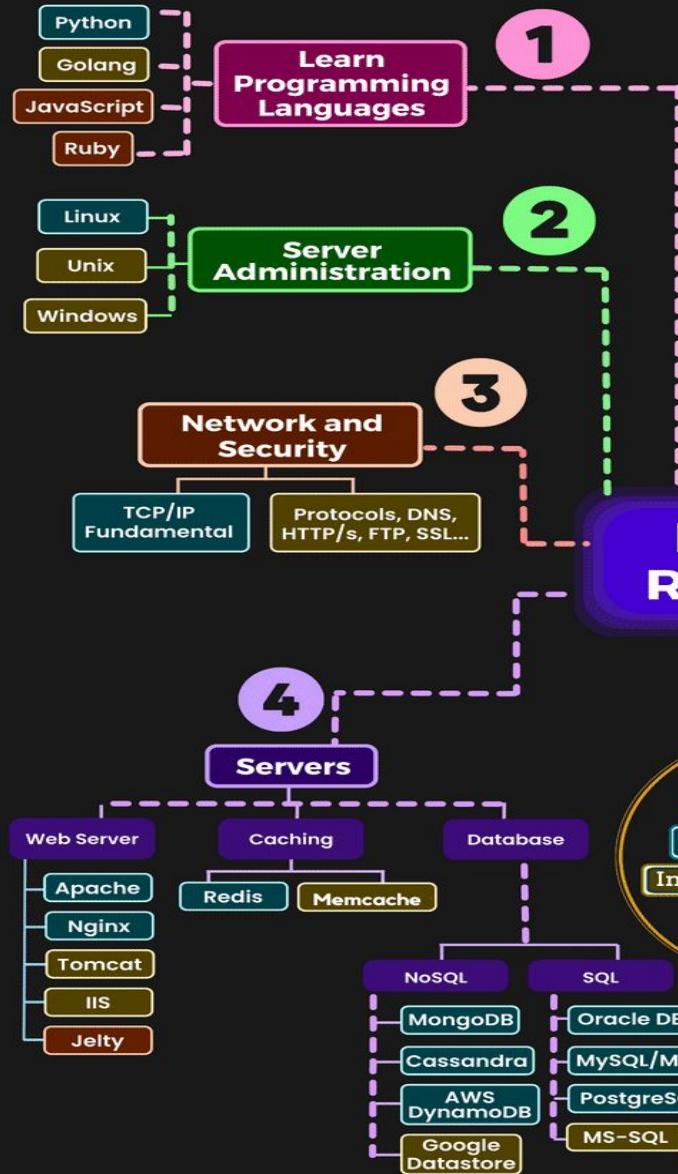
- AWS Certified DevOps Engineer – Professional
- Certified Kubernetes Administrator (CKA)
- Terraform Associate
- Google Cloud Professional DevOps Engineer



# DevOps / SRE Roadmap



Brij Kishore Pandey  
DON'T FORGET TO SAVE



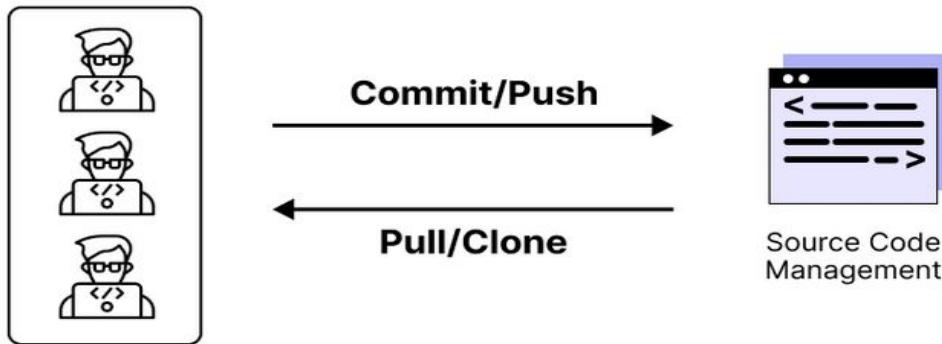
# Devops Lifecycle



# 1 - Continuous Development

Sürekli geliştirme aşaması, yazılımın planlama ve kodlama süreçlerinin gerçekleştirildiği önemli bir aşamadır. İşte bu aşamanın ana noktaları:

- 1. Planlama ve Kodlama**
- 2. Kod Yazımı**
- 3. Kaynak Kod Geliştirme**
- 4. Agile Metodolojisi**
- 5. Parça Parça Geliştirme**



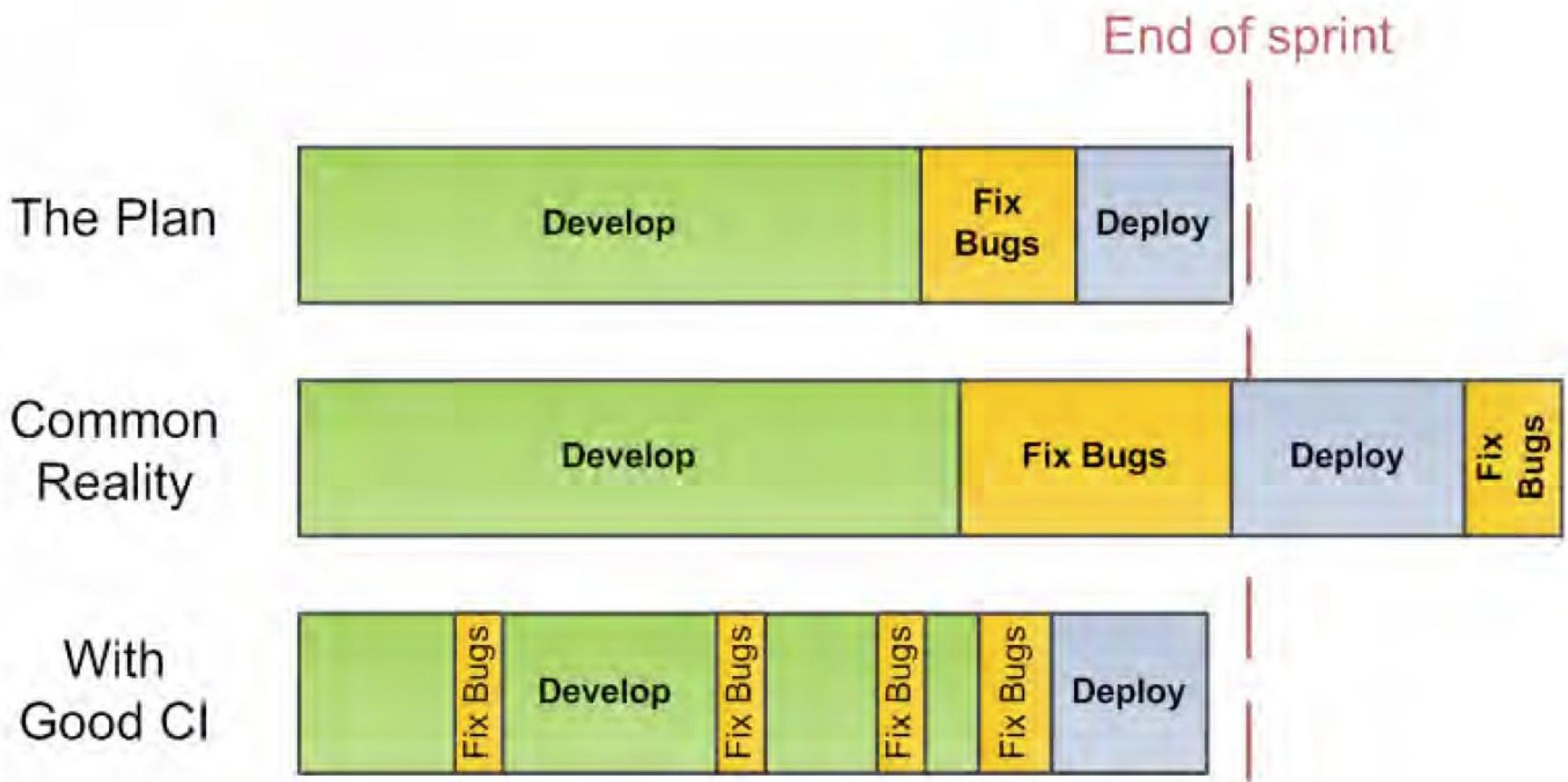
# **Continuous development benefits**

Sürekli geliştirme, yazılım geliştirme süreçlerinde birçok avantaj sunar:

- 1. Yazılım Kalitesinin İyileştirilmesi**
- 2. Değişikliklerin Kolayca Güncellenmesi**
- 3. Hızlı Hata Duzeltmeleri**
- 4. Proje Risk Yönetimi**
- 5. Verimliliğin Artması**
- 6. Zaman ve Kaynak Tasarrufu**

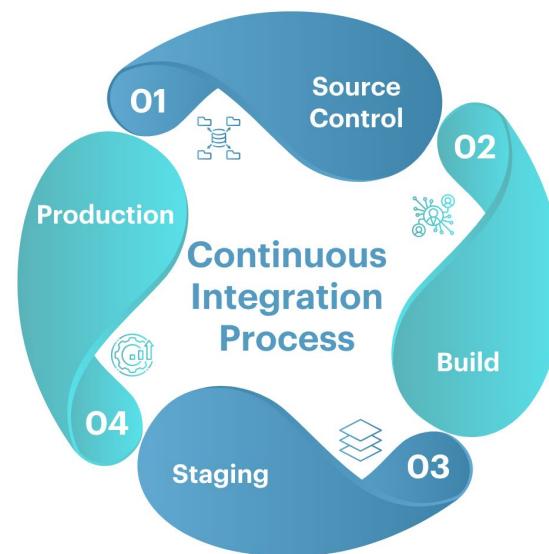
# 2-Continuous Integration

Sürekli entegrasyon, yazılım geliştirme sürecini hızlandırırken, kaliteyi artırmak ve hata oranını azaltmak için kritik bir yöntemdir.



# Continuous integration process

- Source Control (Commit Change)
- Build (Run Build And Unit Tests)
- Staging (Deploy To Test Environment And Run Tests)
- Production (Deploy To Production Environment)



# Benefits of Continuous Integration

Sürekli entegrasyon (CI) uygulamanın sağladığı birçok avantaj bulunmaktadır:

- 1. Hızlı Hata Tespiti**
- 2. Daha Sık Güncellemeler**
- 3. Son Dakika Kaosunu Azaltma**
- 4. Ölçeklenebilirlik**
- 5. Hızlı Geri Bildirim Döngüsü**
- 6. Artan Hesap Verebilirlik ve İletişim**

# **3-Continuous Testing**

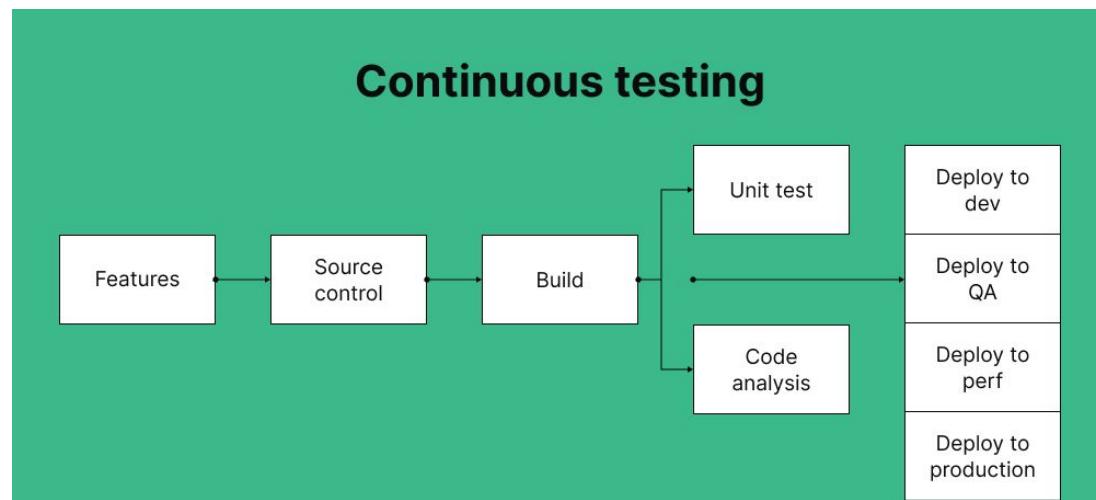
Sürekli test, DevOps yaşam döngüsündeki sürekli entegrasyon aşamasında önemli bir yer tutar ve aşağıdaki aşamaları içerir:

- 1. Esnek Yerleştirme**
- 2. Hata Tespiti**
- 3. Otomatik Testler**
- 4. Kullanıcı Kabul Testi (UAT)**
- 5. Hızlı Geri Bildirim**

# Benefits of Continuous Testing

Sürekli test uygulamanın sağladığı birçok avantaj bulunmaktadır:

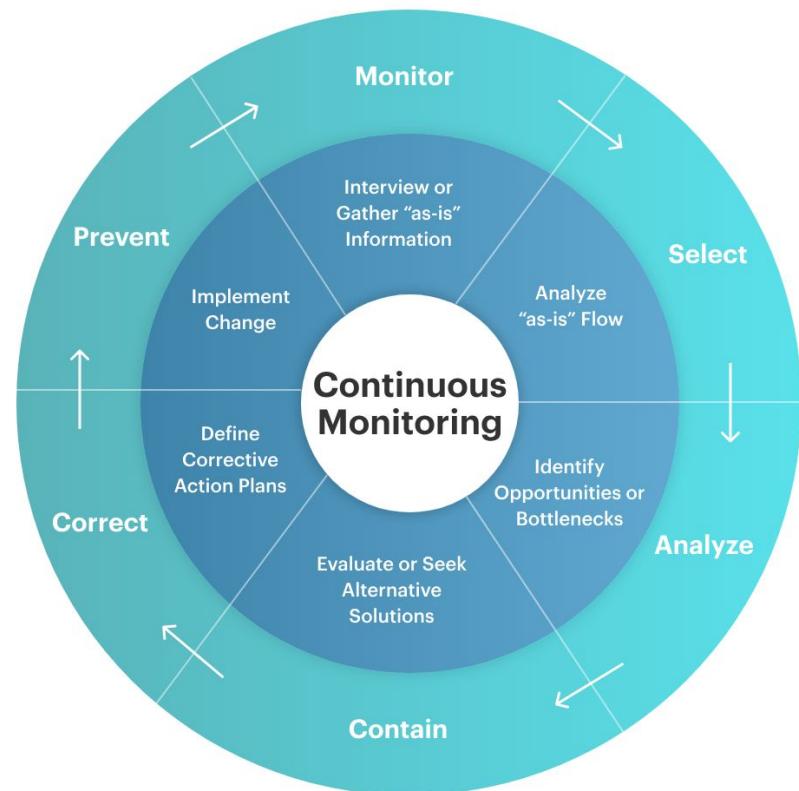
1. **Hızlı ve Güvenilir Geri Bildirim**
2. **Küçük Kod Değişiklikleri**
3. **Daha Hızlı Mean Time to Repair-MTTR (Onarım için Ortalama Süre)**
4. **Hızlı Yazılım Dağılımı**
5. **Erken Hata Tespiti**
6. **Daha Az Kod İnceleme Süresi**
7. **Kaynak Tasarrufu**
8. **Risklerin Azaltılması**



# 4-Continuous Monitoring

Sürekli izleme, DevOps ekibine önemli avantajlar sunarak süreçlerin verimliliğini artırır. İşte bu aşamanın ana noktaları:

- 1. Güvenlik Risklerinin ve Uyumluluk Sorunlarının Hızlı Tespiti**
- 2. Performans İzleme**
- 3. Hata Çözümü**
- 4. Ağ Problemlerinin Yönetimi**
- 5. Sürekli İyileştirme**



# 5-Continuous Feedback

Sürekli geri bildirim aşaması, yazılım geliştirme sürecinin kalitesini artırmak için kritik bir rol oynar. İşte bu aşamanın ana noktaları:

- 1. İyileştirmelerin Analizi**
- 2. Test Edici ve Son Kullanıcı Deneyimleri**
- 3. Geri Bildirimlerin Değerlendirilmesi**
- 4. Yeni Versiyonun Yayınlanması**
- 5. Ekipler Arası Koordinasyon**
- 6. Süreklilik**

# 6-Continuous Deployment

Sürekli dağıtım, son haline getirilmiş kodun üretim sunucularına dağıtıldığı kritik bir aşamadır. İşte bu aşamanın temel noktaları:

- 1. Sonuçlandırılmış Kodun Dağıtıımı**
- 2. Yapılandırma Yönetimi**
- 3. Sunucu Güncellemeleri**
- 4. Otomasyon Araçları**

# 7-Continuous Operations

Sürekli operasyon, DevOps yaşam döngüsünün son ve en kısa aşamasıdır. Bu aşamanın temel hedefleri şunlardır:

- 1. Yazılımın Otomatik Yayıını**
- 2. Performans ve Erişilebilirlik**
- 3. Kesinti Olmadan Çalışma**

“

Your customers probably do not care how the apps they use are deployed, only that they are deployed; preferably with zero-downtime.



## DevOps Phases

### Continuous Planning & Development

- GitLab
- GIT
- TFS
- SVN
- Mercurial
- Jira
- BitBucket
- Trello
- Maven
- Gradle
- Confluence
- Subversion
- Scrum
- Lean
- Kanban

### Continuous Integration

- Jenkin
- Bamboo
- GitLab CI
- TeamCity
- Travis and CircleCI
- Buddy

### Continuous Testing

- JUnit
- Selenium
- JMeter
- Cucumber
- TestSigma
- Microfocus UFT
- TestNG
- Tricentis Tosca
- Jasmine

### Continuous Deployment

- Ansible
- Chef
- Docker
- IBM Urban Code
- Kubernetes
- Puppet
- Go
- Vagrant
- Spinnaker
- ArgoCD

### Continuous Monitoring

- Nagios
- Grafana
- Kibana
- Prometheus
- Logstash
- AppDynamics
- ELK Stack
- New Relic
- Splunk
- Sensu
- PagerDuty

### Customer Feedback

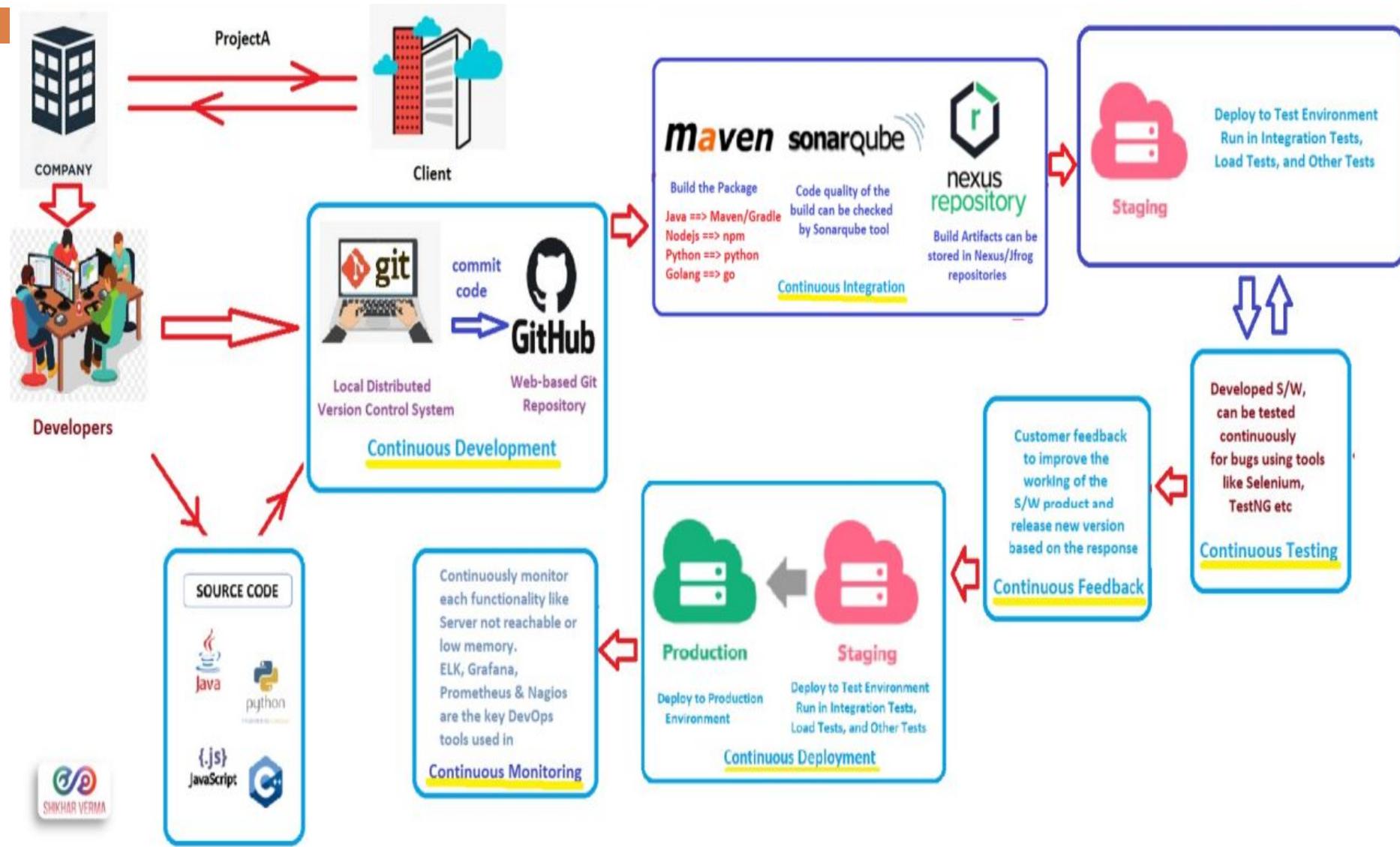
- Webalizer
- W3Perl
- ServiceNow
- Slack
- Flowdock
- Open Web Analytics
- Pendo
- Qentelli's TED

### Continuous Operations

- Kubernetes
- Docker Swarm

## Tools

# 7Cs of the DevOps Lifecycle



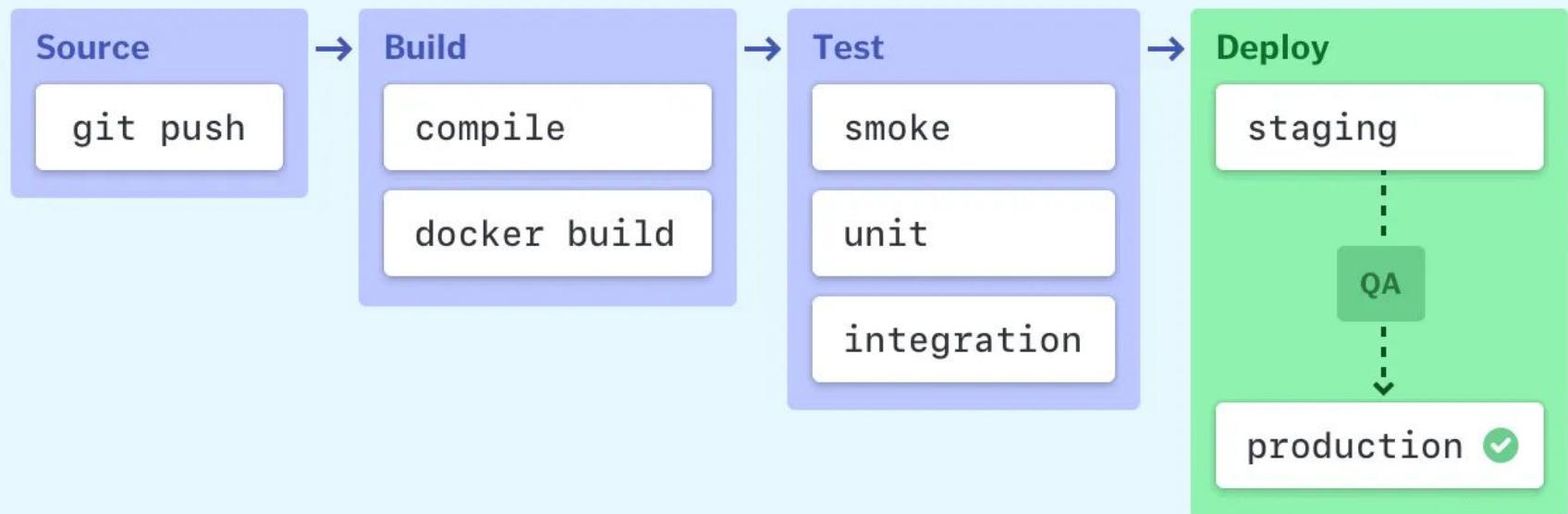
# Why CI/CD?

- **Sık Değişiklikler:** Daha az entegrasyon sorunu.
- **Hatalar Daha Erken Tespit Edilir:** Para tasarrufu sağlar.
- **Son Dakika Kaosunu Önler.**
- **Herkes İçin Şeffaflık.**
- **Üretim Benzeri Ortamlarda Test.**
- **Herhangi Bir Sorun Durumunda Kolayca Geri Alma.**
- **Otomasyon Kültürüünü Teşvik Etme.**
- **DevOps Kültürüünü Güçlendirme.**
- **Geliştiricilerin Hesap Verebilir Olmasını Sağlama ve Sahiplenme Duygusunu Artırma.**

## CI/CD'nin Ana Hedefi:

Hızlı geri bildirim sağlar. Eğer kod tabanına bir hata eklenirse, bu en kısa sürede tespit edilip düzelttilir.

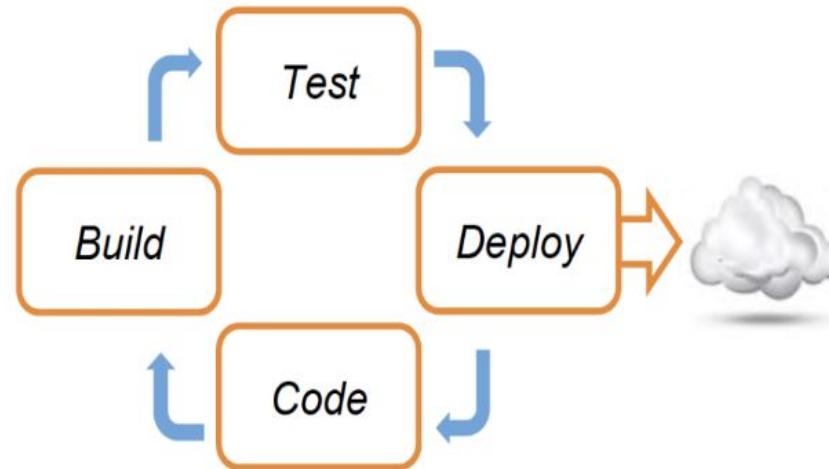
# CI/CD Pipeline



# Continuous Delivery & Continuous Deployment

## Continuous Delivery (CD)

Sürekli teslimat, yazılımın her zaman üretim ortamına geçmeye hazır hale getirilmesini ifade eder.

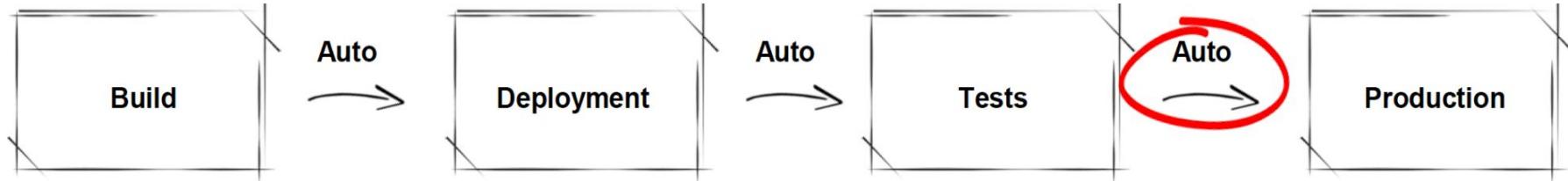


## Continuous Deployment

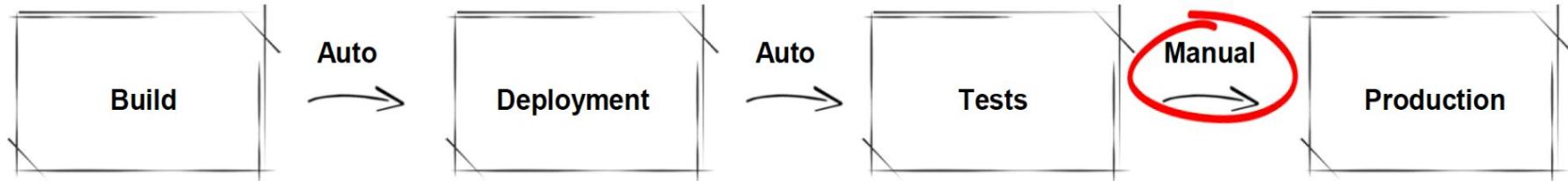
Sürekli dağıtım, her yeni kod güncellemesinin otomatik olarak üretim ortamına aktarılması sürecidir.

# Continuous Delivery & Continuous Deployment

## Continuous Deployment

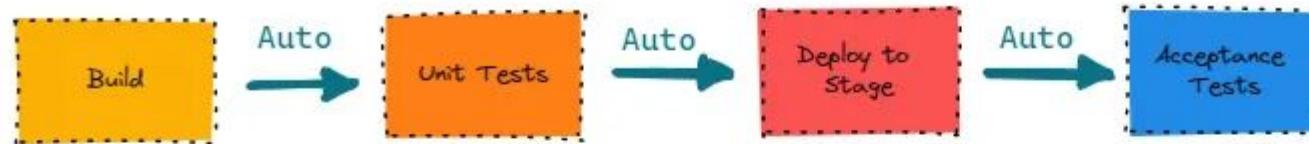


## Continuous Delivery



# Continuous (Integration Delivery Deployment)

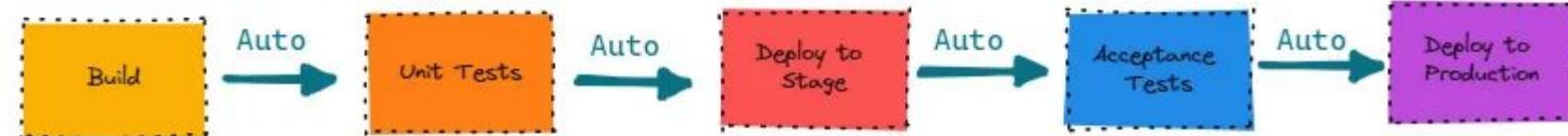
## Continuous Integration



## Continuous Delivery

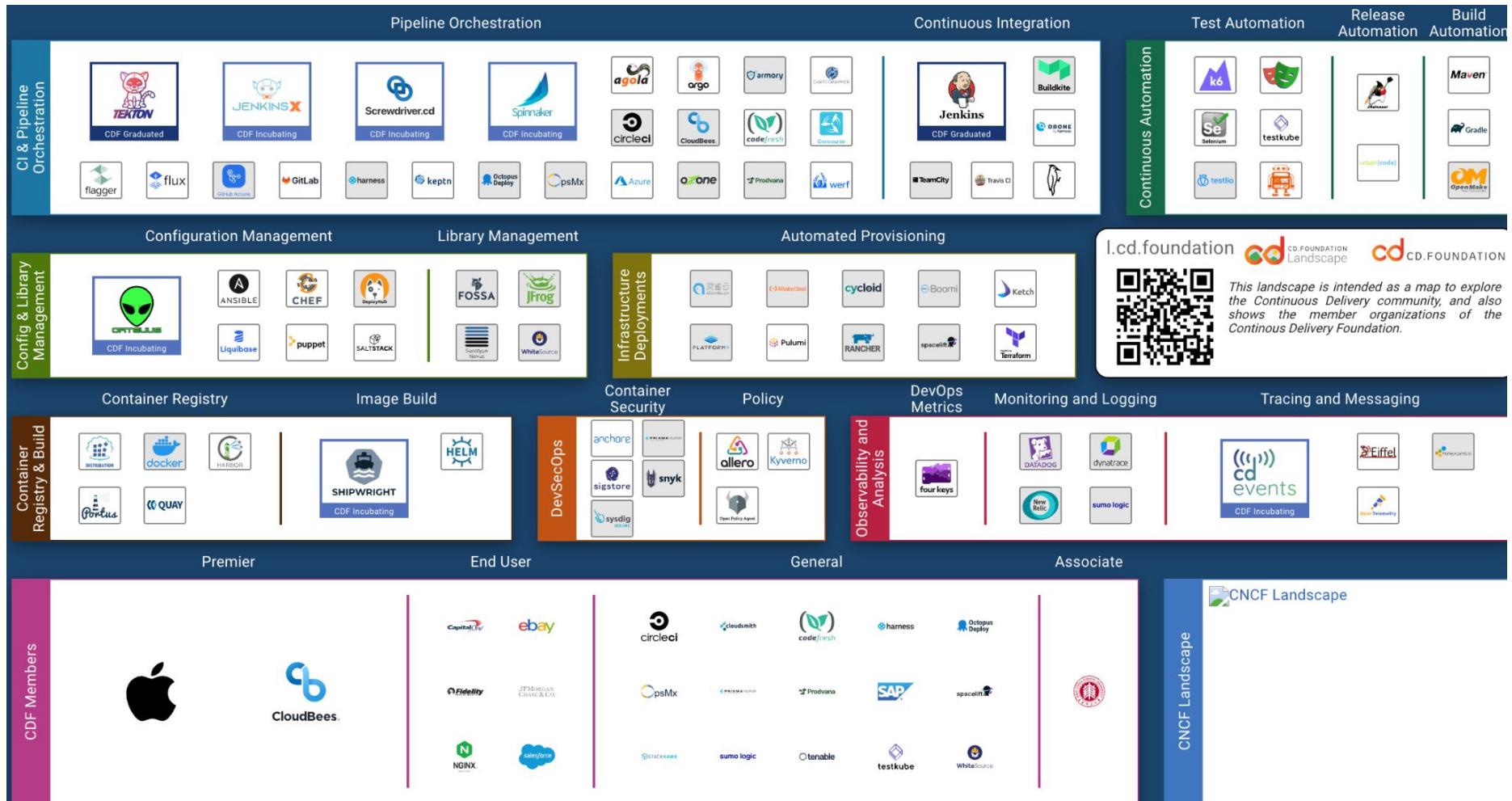


## Continuous Deployment



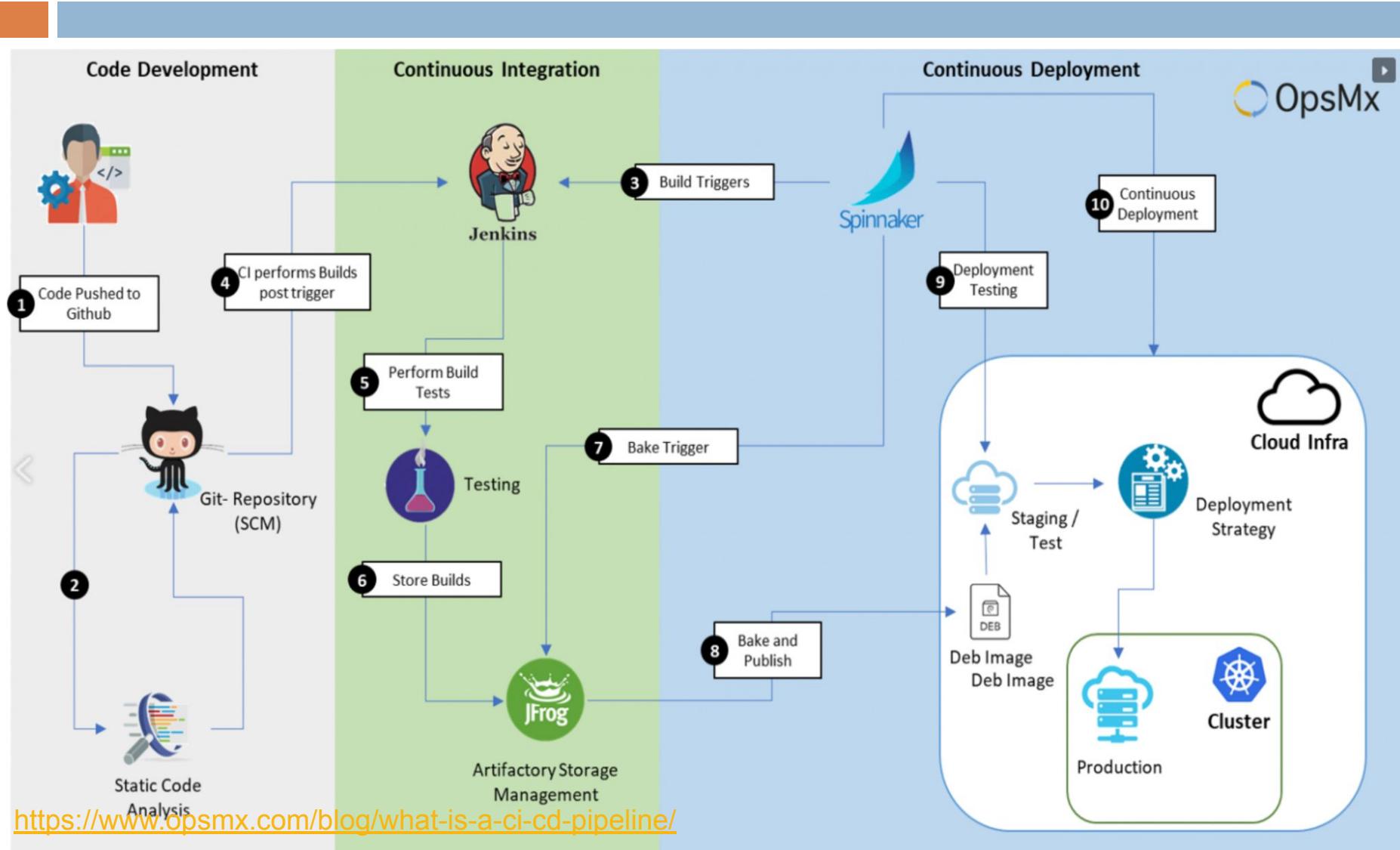
Continuous Delivery'de deploy işlemi manuel olurken  
Continuous Deployment'da deploy işlemi otomatik yürütülmektedir.

# Continuous Delivery Landscape

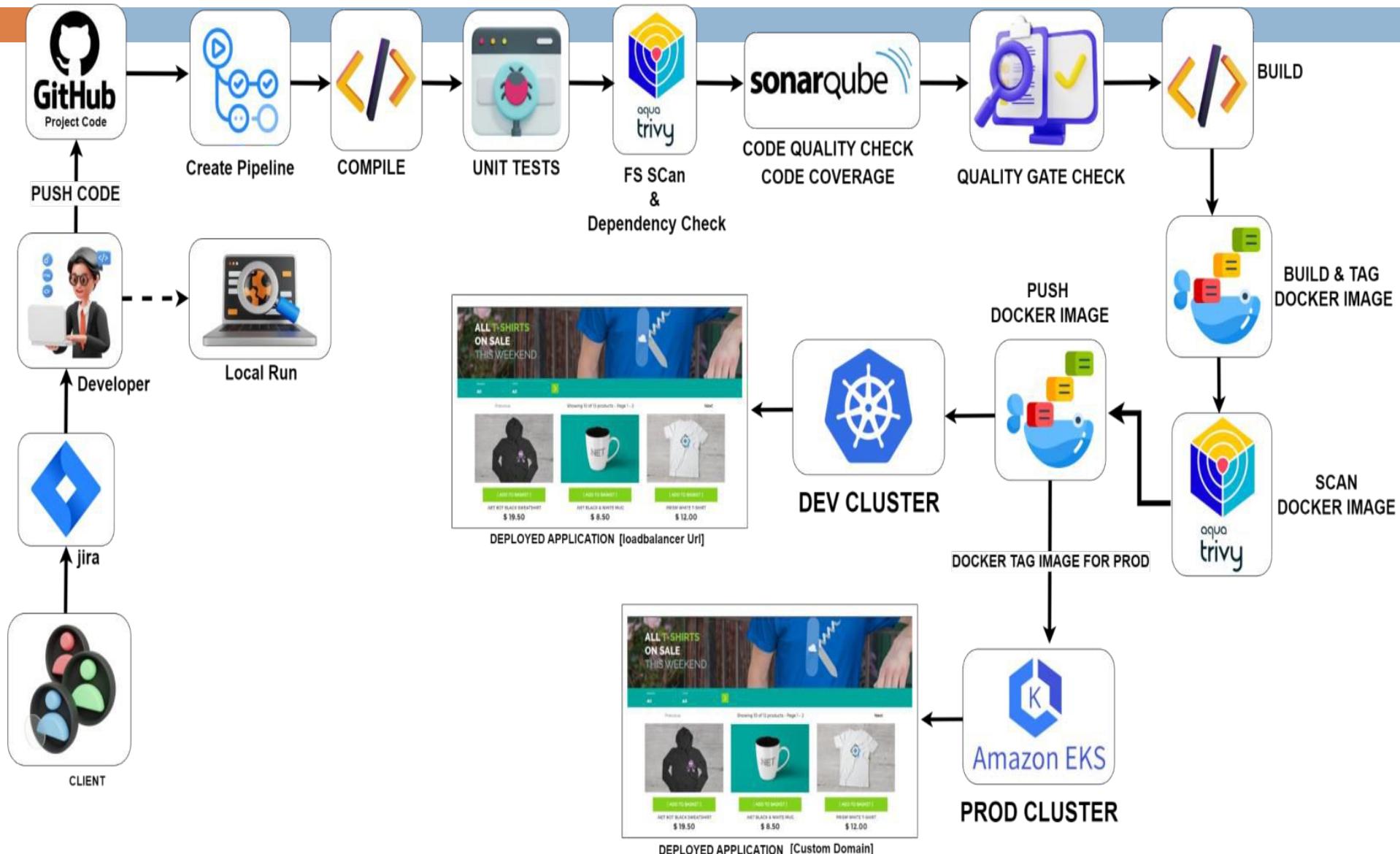


<https://landscape.cd.foundation/>

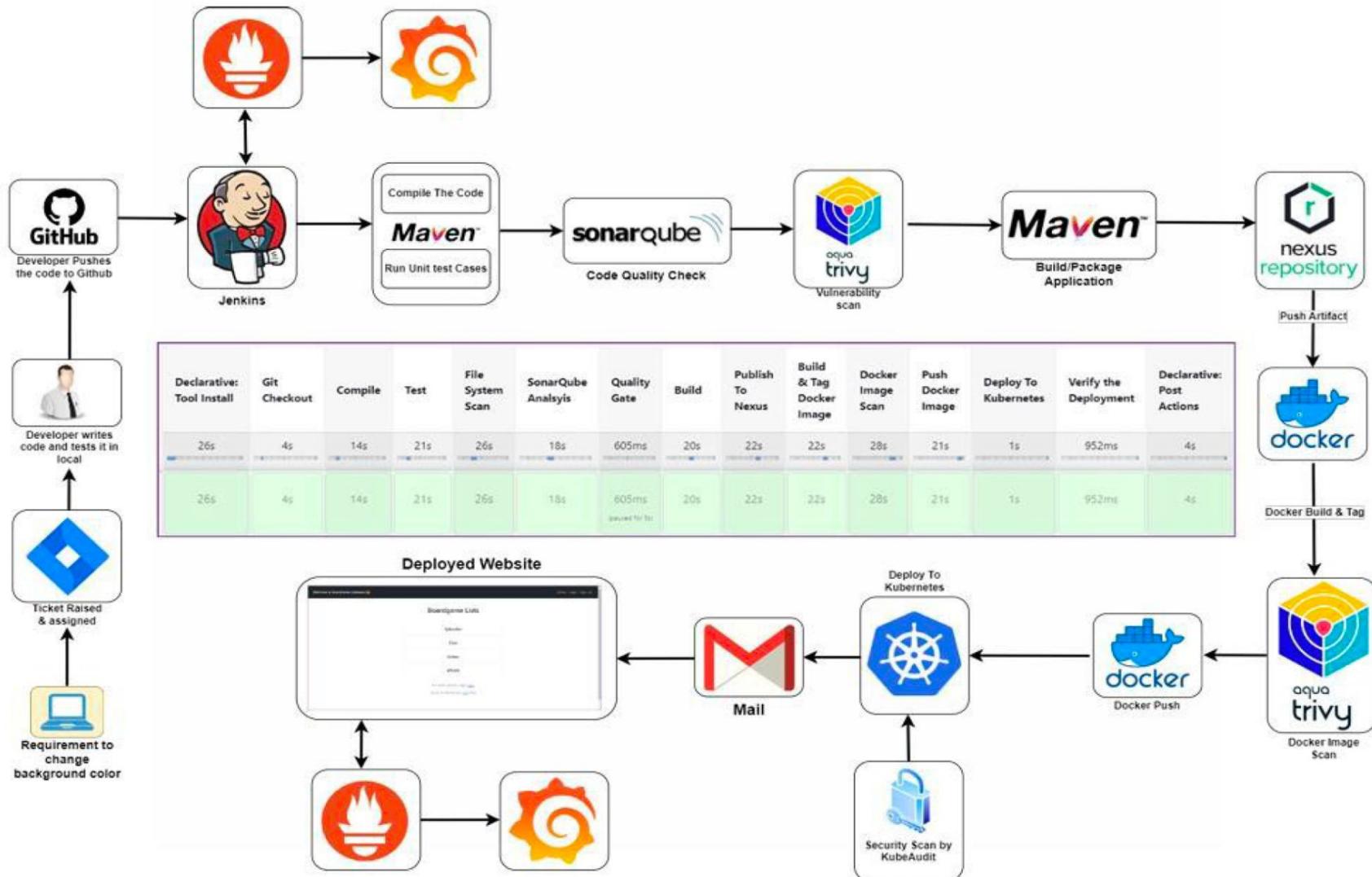
# CI/CD



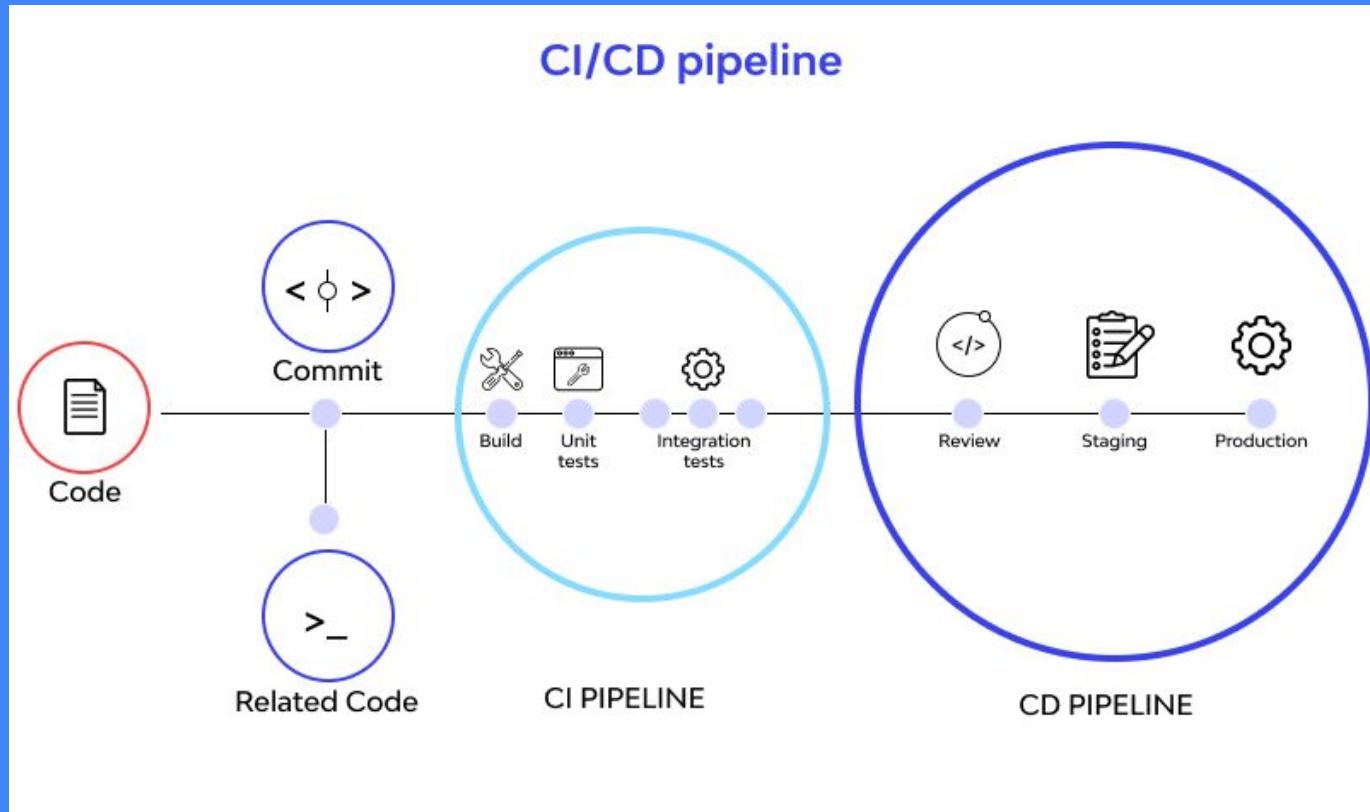
# CI/CD DevOps Project



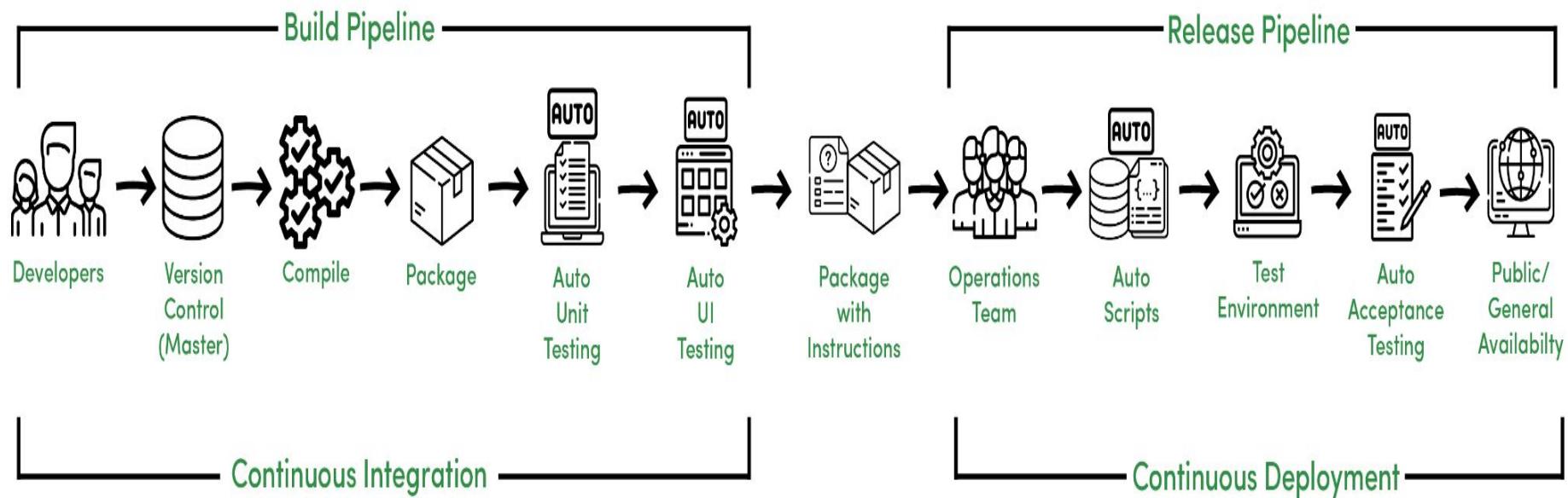
# CI/CD DEVOPS PIPELINE PROJECT



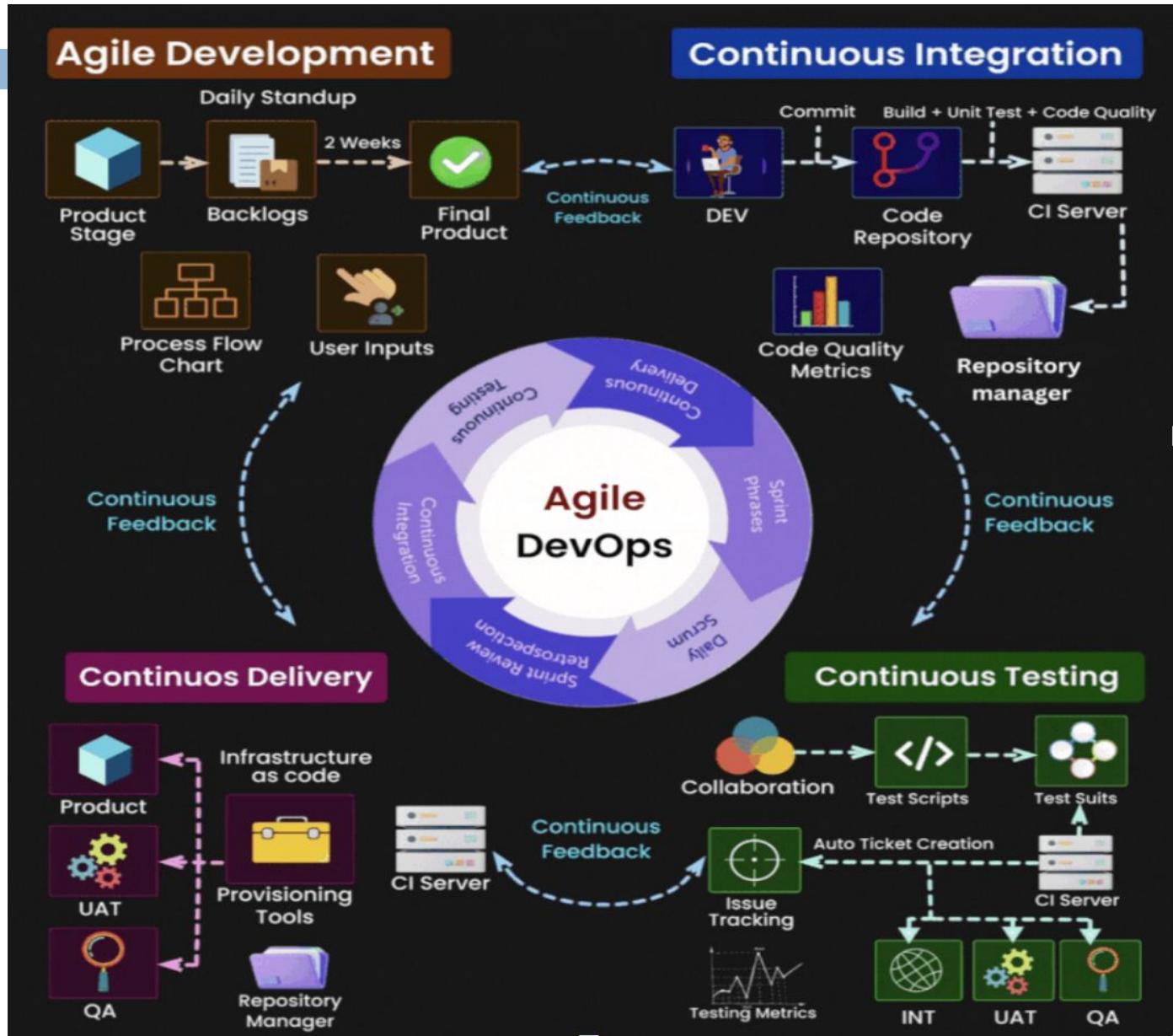
# Şirketlerde Kod Nasıl Yazılıyor



# Şirketlerde Kod Nasıl Yazılıyor



# Agile with DevOps



# Şirketin Yeni Özellik Geliştirme Süreci: Agile ve DevOps Entegrasyonu

Bir yazılım geliştirme şirketinde, yeni bir özellik talebi geldi. Şirket, **Agile** ve **DevOps** metodolojilerini entegre ederek bu özelliği geliştirme ve teslim etme sürecini başlattı. Bu süreçte çeşitli roller, araçlar ve adımlar yer alıyor. İşin en başından, teslimatına kadar olan bu döngüde yer alan adımları ve araçların nasıl kullanıldığını birlikte inceleyelim.



# Projede Görev Alan Kişiler ve Rolleri

## 1. Proje Yöneticisi (Scrum Master): Ahmet

- **Görevi:** Proje akışını ve takımın işbirliğini yönetir. Sprint planlamaları yapar, günlük Scrum toplantılarını düzenler ve ekip üyelerinin karşılaşduğu engelleri kaldırır.

## 2. Ürün Sahibi (Product Owner): Elif

- **Görevi:** Müşteriden gelen gereksinimleri toplar ve kullanıcı hikayeleri yazar. Backlog yönetimini yapar, görevlerin önceliklendirilmesini sağlar ve müşteri ile sürekli iletişimde kalarak doğru yönlendirmeler yapar.

## 3. Geliştirici (Developer): Ali

- **Görevi:** Flask tabanlı yeni mikroservis özelliğini geliştirir. Kod yazma, birleştirme, test etme ve Docker image oluşturma işlemlerini yapar.

## 4. DevOps Mühendisi (DevOps Engineer): Mehmet

- **Görevi:** Altyapıyı hazırlar, Kubernetes ve Docker yapılandırmalarını yönetir. Jenkins pipeline'ını yazar ve uygulamanın Kubernetes'e dağıtımını sağlar. CI/CD sürecini yönetir.

## 5. Yazılım Testçisi (Tester): Zeynep

- **Görevi:** Ali'nin geliştirdiği yeni özelliği test eder. Otomatik test senaryoları yazar ve bunları Jenkins pipeline'a entegre eder. Sonuçları raporlar.

## 6. Müşteri (Customer): Murat

- **Görevi:** Geliştirilen yeni özelliği test eder, geribildirim verir ve kabul testlerini gerçekleştirir.

# 1-Planlama ve Gereksinim Toplama (Sprint Planning & Backlog)

## Agile aşaması:

- **Proje Yöneticisi (Scrum Master): Ahmet**

Ahmet, ekibi toplayarak müşteri ihtiyaçlarına göre projenin yol haritasını belirler. Ürün sahibi Elif ile işbirliği yaparak sprint planlamasını yapar ve backlog'dan geliştirilmesi gereken yeni özellikleri seçer.

- **Ürün Sahibi (Product Owner): Elif**

Elif, müşteri Murat'tan aldığı gereksinimlere göre kullanıcı hikayeleri yazar ve sprint backlog'una ekler. Elif, bu hikayeleri önceliklendirir ve Ahmet ile sprint sürecinde yapılacak işleri netleştirir.

**Sprint Hedefi:** Flask tabanlı yeni bir mikroservis geliştirmek ve Kubernetes ortamına deploy etmek.

## DevOps Aşaması:

- Bu aşama, DevOps'un Continuous Planning aşamasına denk gelir. Altyapı ve yazılım gereksinimleri belirlenir, yazılımın sürekli entegrasyonu ve teslimatı için gerekli adımlar planlanır.

**Araçlar:** Jira (Sprint planlaması için).

## 2-Geliştirme Süreci(Sprint Execution & Continuous Integration)

### Agile Aşaması:

- **Geliştirici Ali**, sprint boyunca belirlenen yeni özellik üzerinde çalışır. Flask kullanarak mikroservisi geliştirir ve bu servisin Docker container'da çalışabilmesi için bir **Dockerfile** oluşturur.
- Ali, kodu **Git** kullanarak kendi dalına (branch) commit eder. Sprint boyunca ekipten geri bildirimler alır ve kodunu sürekli olarak iyileştirir.
- Geliştirme süreci tamamlandığında, Ali ana dal (main branch) ile birleştirme için **pull request (PR)** açar. PR, takım tarafından gözden geçirilir ve onaylanırsa ana repository'ye merge edilir.

### DevOps Aşaması:

- **DevOps Mühendisi Mehmet**, Jenkins üzerinde bir **pipeline** oluşturur. Pipeline, kodun sürekli entegrasyonu (**CI**) ve sürekli teslimatı (**CD**) işlemlerini otomatize eder. **Jenkinsfile** yazılır ve GitHub repository'sine eklenir.
- Her kod değişikliği (commit) yapıldığında Jenkins pipeline otomatik olarak çalıştırılır, kod test edilir, Docker image oluşturulur ve bir sonraki aşamaya hazırlanır. Bu süreçte **Continuous Integration (CI)** devreye girer.
- **Araçlar**: Git, GitHub, Jenkins, Docker,

### 3-Kod Kalitesi Analizi (**SonarQube & Trivy**)

#### Agile Aşaması:

- **Sprint Execution** devam ederken geliştiriciler ve testçiler kodu sürekli olarak gözden geçirir.

#### DevOps Aşaması:

- **Jenkins pipeline** çalıştırıldığında, **SonarQube** ile kod kalitesi analizi yapılır. Mehmet, kodun güvenliğini kontrol etmek için **Trivy** kullanarak Docker image'ını tarar. **Continuous Testing** devreye girer.

**Araçlar:** SonarQube, Trivy.

# 4-Altyapı Yönetim ve Kubernetes Deployment

## Devops Aşaması

- **Mehmet (DevOps Engineer):**
  - Mehmet, **Terraform** kullanarak AWS üzerinde Kubernetes altyapısını yönetir ve gerekli Kubernetes cluster'ını oluşturur. **kubectl** komutlarıyla uygulamayı Kubernetes'e dağıtır. **Argo CD** ile GitOps modeline dayalı sürekli teslimat (**CD**) sağlanır.

**Araçlar:** Kubernetes, Terraform, Argo CD.

## 5-Test Süreci (Testing Phase)

### Agile Aşaması:

- **Yazılım Testçisi (Tester): Zeynep**

Ali'nin geliştirdiği yeni mikroservis, Zeynep tarafından test edilir. Jenkins pipeline'a entegre edilen otomatik test senaryoları çalıştırılır ve sonuçlar raporlanır.

- **Müşteri (Customer): Murat**

Müşteri Murat, geliştirilmiş olan yeni özelliği kabul testlerinden geçirir. Murat'ın onayı ile ürün son kullanıcıya sunulur.

- **Araçlar:** Jenkins, Selenium, TestNG

## 6-Monitoring ve Log yönetimi(Operasyonel Süreç)

### Devops Aşaması:

- **Mehmet**: Uygulamanın üretim ortamında çalıştığından emin olmak için izleme araçlarını yapılandırır.
  - **Prometheus** ile uygulamanın metrikleri toplanır.
    - `kubectl apply -f prometheus.yaml`.
  - **Grafana**: Toplanan metrikler görselleştirilir ve dashboard oluşturulur.
    - `kubectl apply -f grafana.yaml`.
  - **ELK Stack**: Loglar toplanır ve Kibana üzerinde görselleştirilir.

**Araçlar**: Prometheus, Grafana, ELK Stack.

**Uygulama performansı izlenir ve gerekiğinde anında müdahale yapılır.**

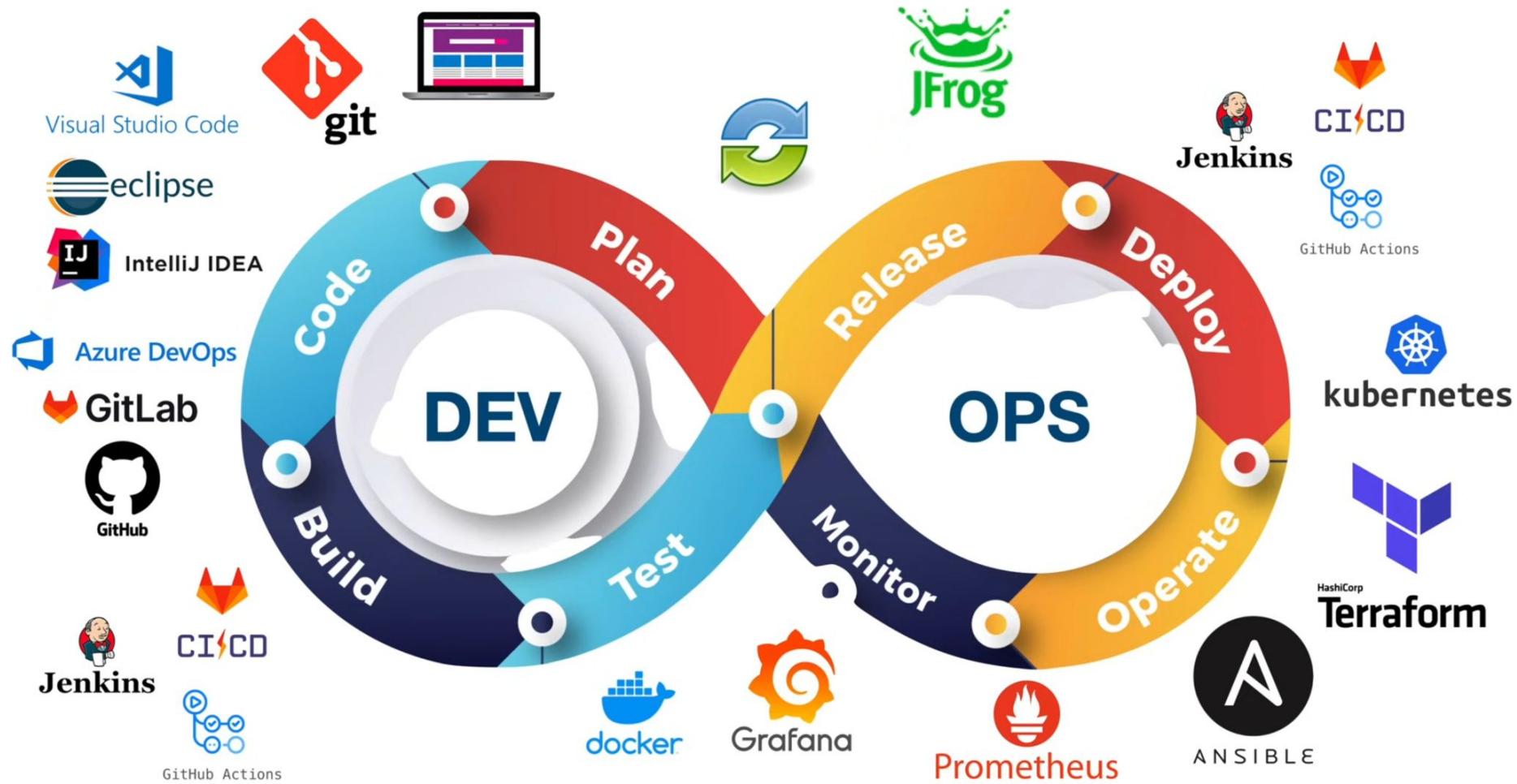
## 7-Müşteri Onayı ve Sonuç

### Agile Aşaması:

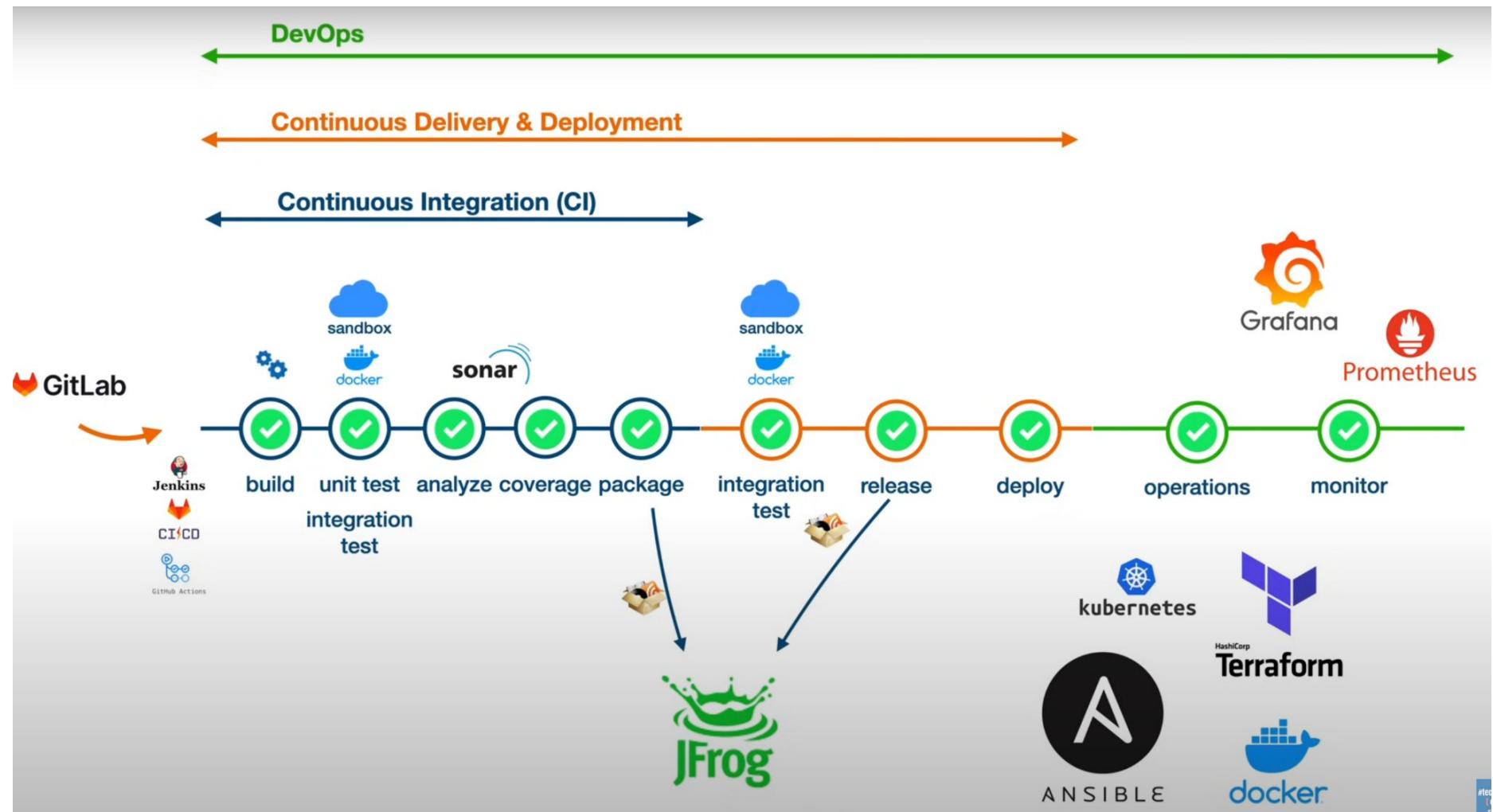
- **Müşteri (Customer):** Yeni özelliği kullanarak test eder ve kabul testi yapar.
- **Ali ve Mehmet:** Müşteri geri bildirimi doğrultusunda iyileştirme yapar ve son dokunuşları gerçekleştirir.

**Müşteri onayı alındıktan sonra yeni özellik başarılı bir şekilde devreye alınır. Proje tamamlanır ve sprint kapanır.**

# Devops Tools

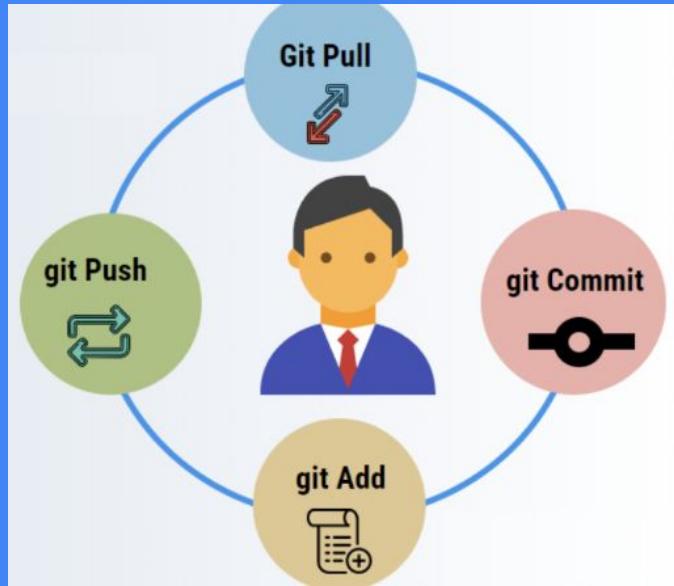


# Şirketlerde Kod Nasıl Yazılıyor





# Git & Github

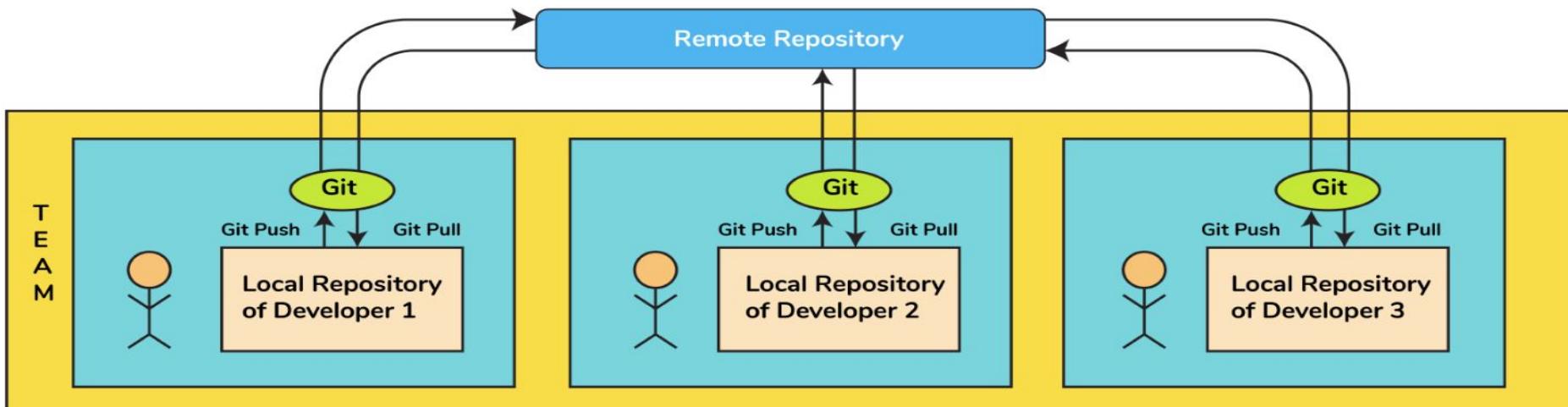


# Git & Github

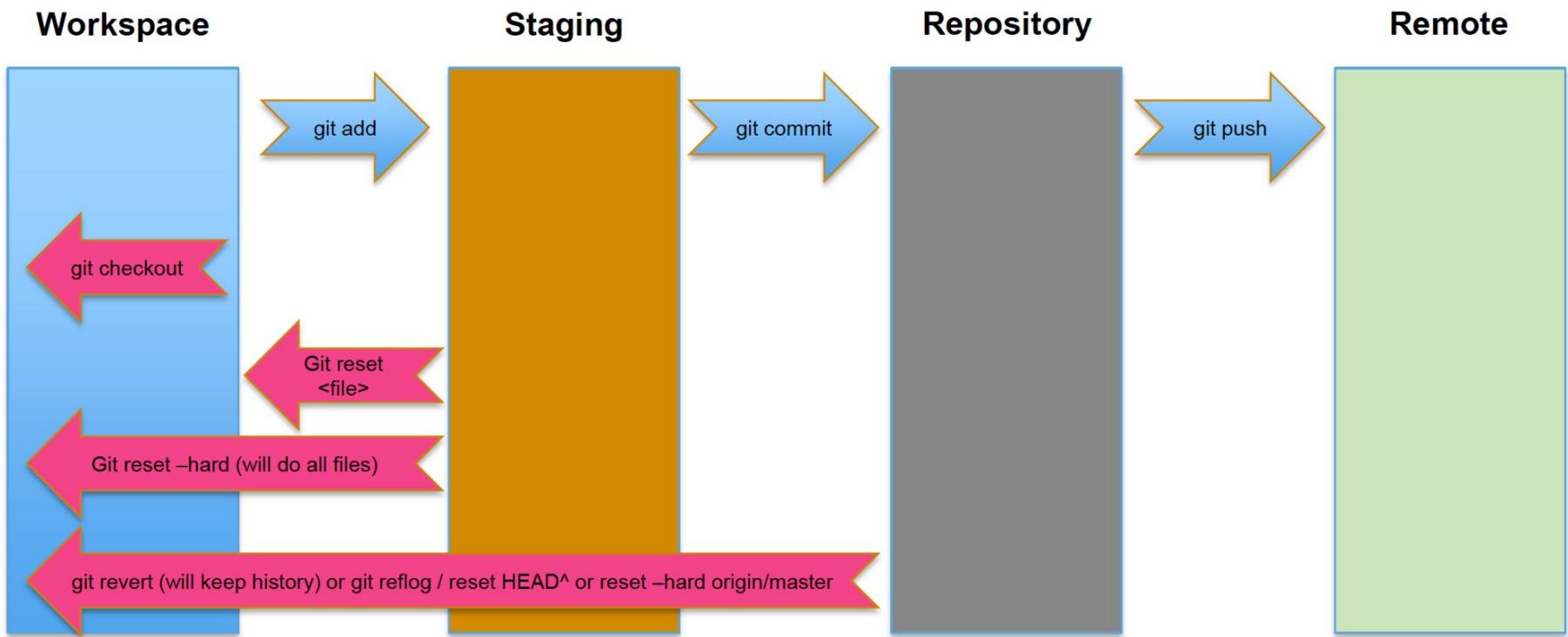
- **Getting Started with Git**
- **Understanding GitHub**
- **Git Commands**
- **Collaborating on GitHub**
- **Working with SSH Keys and Personal Access Tokens**
- **Continuous Integration with GitHub**

# Git Nedir?

- Git, yazılım geliştirme sürecinde koddaki değişiklikleri takip etmek için kullanılan ücretsiz ve açık kaynaklı bir **versiyon kontrol sistemi**dir.
- Geliştiricilerin, **farklı geliştirme branch (dallarını)** yönetmesine olanak tanıyarak paralel çalışma süreçlerini mümkün kılar.
- Ekip üyeleri arasında **iş birliğini kolaylaştırır ve hız ve verimlilik** için tasarlanmıştır.
- **Çoklu platform desteği** sunar ve herhangi bir IDE/Entegre eklientilerle uyumludur.



# Git: Areas



# GIT Key Terminology

**Git Repository:** Git deposu, dosyalar, geçmiş ve yapılandırmaları içeren, GIT tarafından yönetilen bir alanı ifade eder.

**Working Directory:** Canlı dosyaların bulunduğu, GIT'in henüz takip etmediği "izlenmeyen" alandır.

**Staging Area:** Git'in dosyalardaki değişiklikleri takip etmeye ve kaydetmeye başladığı alandır.

**Git Directory:** "Yerel depo" olarak da bilinen, git'in her şeyi kaydettiği `.git` klasöründür.

**Remote Repository:** GitHub gibi bir kod barındırma hizmetinde ya da iç sunucuda depolanan uzaktaki git deposudur.

**Branch in Git:** Git'teki bir branch, projedeki ana parça dokunmadan yeni bir özellik ya da değişiklik geliştirmeye devam etmeyi sağlar.

# Github Nedir?

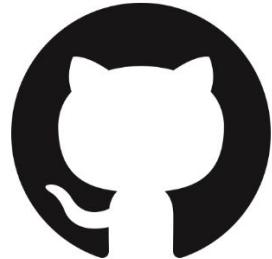
**GitHub**, dünyanın en büyük kaynak kod barındırma platformudur ve 2018'den beri Microsoft'a aittir. Git ile entegre çalışan GitHub, geliştiricilerin projelerini yönetmelerini, paylaşmalarını ve işbirliği yapmalarını sağlar. Açık kaynak projelerden özel projelere kadar geniş bir yelpazede kullanım sunar.

- 

Web-based hosting service for version control using Git.  
The largest source code in the world



# Lets install git & create GitHub account



**Linux based:** \$ sudo apt-get install git

**Windows:** Download from <https://git-scm.com/>

<https://github.com/>

Sign Up

**Mac:** \$ brew install git Or download from <https://git-scm.com/>

## Initial configurations:

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

## Check your configurations:

```
$ git config -l
```

# Git installation / configuration files

INSTALLATION DIRECTORY



CODE REPOSITORY



CONFIGURATION DIRECTORY



[user]

name = Nir Koren  
email = nirk@liveperson.com

[merge]

tool = p4merge

...

# Basic Git Commands

**git init**: Yeni bir Git deposu başlatır.

- Projenizi bir Git deposu olarak başlatmak için kullanılır.

**git clone <url>**: Mevcut bir depoyu URL'den klonlar.

- Uzak bir Git deposunu yerel bilgisayarınıza kopyalar.

**git add <file>**: Dosyada yapılan değişiklikleri bir sonraki commit için sahneler.

- Belirtilen dosyayı sahneye ekler, yani commit edilmeye hazır hale getirir.

**git commit -m "message"**: Sahnelenmiş değişiklikleri mesajla birlikte commit eder.

- Yaptığınız değişiklikleri bir mesaj ile kaydeder.

**git status**: Çalışma dizininizin ve sahneleme alanınızın durumunu gösterir.

- Hangi dosyaların değiştiğini, sahnelendiğini ve commit edilmeye hazır olduğunu gösterir.

**git push**: Yerel değişikliklerinizi uzak bir depoya gönderir.

- Commit'lerinizi GitHub gibi uzak bir depoya göndererek projenizi güncellersiniz.

# Working with Remotes

**Remotes:** GitHub gibi platformlarda barındırılan depoların URL'leridir.

**git remote add <name> <url>**: Yeni bir uzak depo ekler.

- Belirttiğiniz isme ve URL'ye sahip bir uzak depo eklemek için kullanılır.

**git fetch**: Uzak depodan değişiklikleri alır.

- Uzak depodaki güncellemeleri indirir, ancak yerel dalınıza uygulamaz.

**git pull**: Uzak depodan değişiklikleri çeker ve yerel dalınıza birleştirir.

- Uzak depodaki güncellemeleri indirir ve otomatik olarak mevcut dalınıza entegre eder.

# Branching and Merging

**Branching Yönetimi:** Branching, projenizin farklı sürümleri üzerinde aynı anda çalışmanızı olanak tanır.

**git branch <branch-name>**: Yeni bir dal oluşturur.

- Belirttiğiniz isimle yeni bir dal oluşturmak için kullanılır.

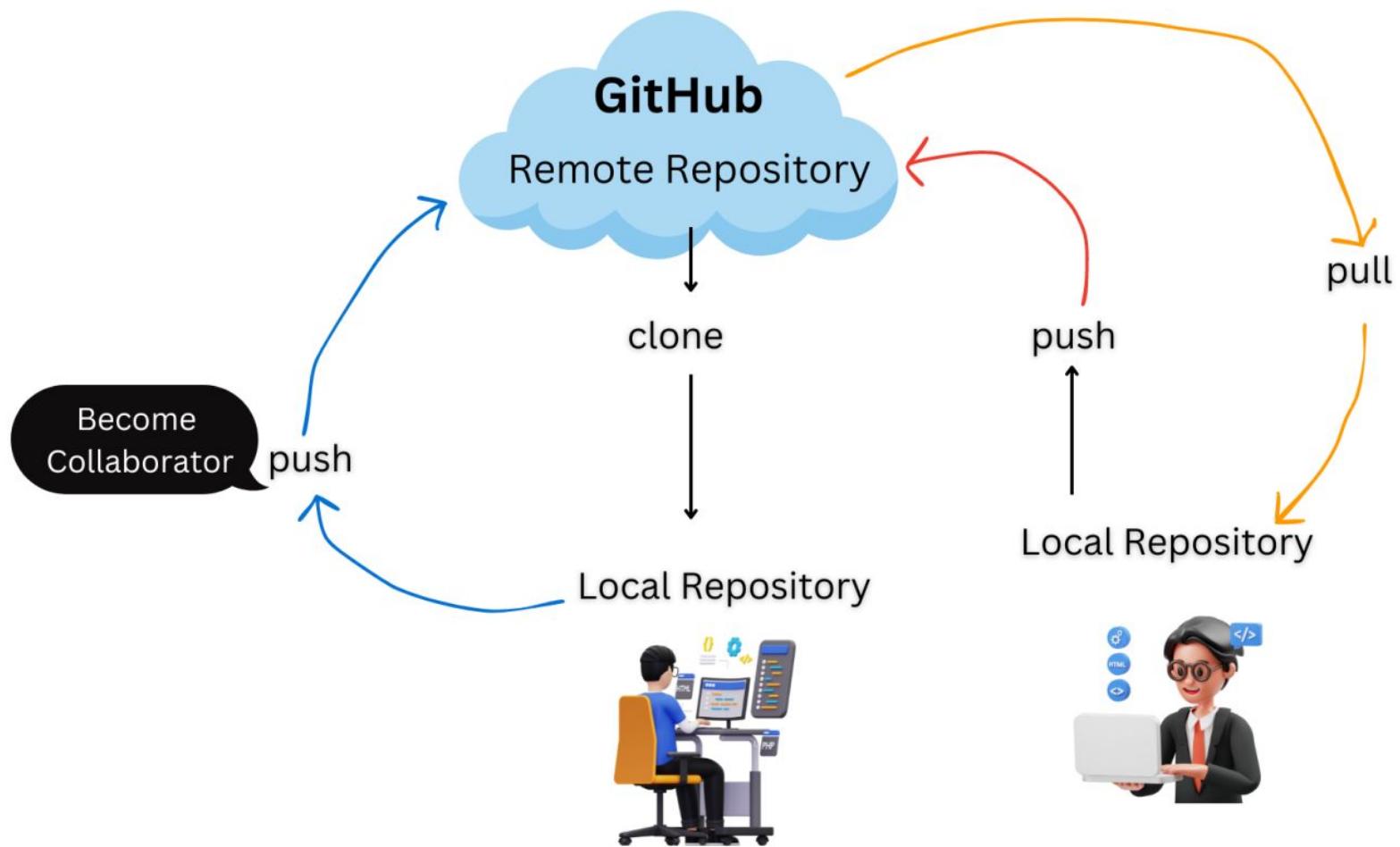
**git checkout <branch-name>**: Belirtilen dala geçiş yapar.

- Çalışmak istediğiniz dala geçiş yapmanızı sağlar.

**git merge <branch-name>**: Belirtilen dalı mevcut dala birleştirir.

- Farklı bir dalda yapılan değişiklikleri mevcut dalınıza entegre eder.

# Collaborating on GitHub



# Forking Repositories & Pull Requests

**Forking**, başkasının deposunun kendi GitHub hesabınız altında bir kopyasını oluşturur. Bu kopyada değişiklik yapabilir, ancak orijinal depoyu değiştiremezsiniz.

## Pull Request'ler:

1. Katkıda bulunmak istediğiniz depoyu fork ve klonlayın.
2. Değişiklikleriniz için yeni bir dal (branch) oluşturun.
3. Değişikliklerinizi yapın ve bunları fork'ladığınız depoya push (yükleyin).
4. Kendi dalınızdan orijinal depoya bir pull request (çekme isteği) açın.

# Code Reviews and Collaboration

**Collaboration:** birden fazla kişinin veya ekibin bir proje üzerinde birlikte çalışması anlamına gelir. GitHub'da bu süreç şunları içerir:

- **Team Members:** Geliştiriciler projeye değişiklik yaparak katkıda bulunabilirler.
- **Branch Management:** Geliştirme, bağımsız çalışmalar için ayrı dallarda gerçekleşir.
- **Pull Requests:** Değişiklikler, inceleme ve onay sonrası ana projeye dahil edilir.
- **Comments and Discussions:** Ekip üyeleri kod üzerinde yorum yapar ve geri bildirimde bulunur.
- **Version Control:** GitHub değişiklikleri takip eder ve gerekirse geri almayı kolaylaştırır.

# Code Reviews

## Code Reviews Steps

- **Code Submission:** Kod, projeye eklenmesi için gönderilir.
- **Review:** Diğer ekip üyeleri kodu gözden geçirir
- **Feedback:** Kod hakkında yorum ve öneriler yapılır.
- **Revisions or Corrections:** Geri bildirime göre kodda değişiklikler yapılır.
- **Approval and Merging:** Kod, onaylandıktan sonra ana projeye dahil edilir.

# Working with SSH Keys and Personal Access Tokens



## GitHub için SSH Anahtarlarının Ayarlanması

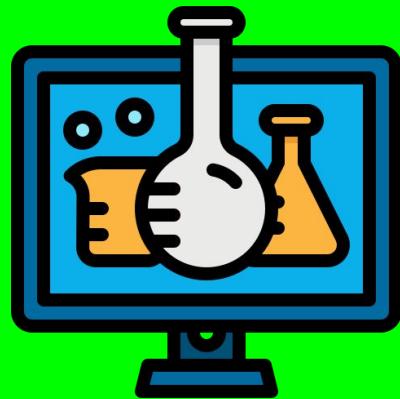


- SSH Anahtar Çifti Oluşturma:** `ssh-keygen` komutunu kullanarak bir SSH anahtar çifti oluşturun ve yönergeleri izleyin.
  - Bu, sizin ve GitHub arasındaki güvenli bağlantıyı sağlar.
- Açık Anahtarını GitHub Hesabınıza Ekleme:** Oluşturduğunuz açık anahtarları GitHub hesabınızdaki "SSH and GPG keys" bölümüne ekleyin.
  - Bu adım, GitHub'ın sizin kim olduğunuzu tanımasını sağlar.
- Bağlantıyı Test Etme:** `ssh -T git@github.com` komutunu kullanarak bağlantıyı test edin.
  - Başarılı bir bağlantı, GitHub ile doğru bir şekilde iletişim kurduğunu gösterir.

## Kişisel Erişim Belirteçlerinin (PAT) Yapılandırılması

- PAT Oluşturma:** GitHub hesabınızdaki "Developer settings" bölümünden bir Kişisel Erişim Belirteci (PAT) oluşturun.
  - Bu PAT, Git işlemlerinde kimlik doğrulama için kullanılacaktır.
- Token'ı Şifre Yerine Kullanma:** Git işlemlerini gerçekleştirirken şifre yerine bu Token'ı kullanın.
  - Bu, kimlik doğrulama işlemlerini daha güvenli hale getirir.

# Git & Github Exercises



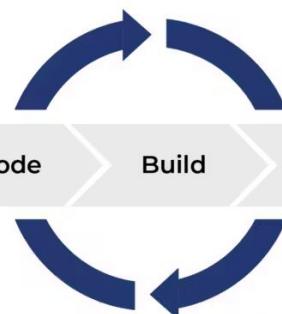
# Github Actions



# GitHub Actions Nedir?

**GitHub Actions**, GitHub üzerinde süreçleri otomatikleştirmenizi sağlayan bir özellikleştir. Yazılım geliştirme yaşam döngüsündeki birçok adımı (CI/CD, test, dağıtım, kod kontrolü, güvenlik denetimleri, bildirimler, vb.) yönetmek için kullanılır. GitHub Actions, bir iş akışını ("workflow") tanımlayarak belirlenen olaylara göre bu iş akışlarını tetikler ve işlemleri otomatik olarak yürütür.

## CI/CD



# GitHub Actions Mimarisi

**GitHub Actions**, belirli olaylara (trigger) yanıt vererek başlar. Genellikle **push**, **pull request** veya zamanlayıcı bazlı (cron) olaylarla tetiklenir.

**Events** : İş akışlarını başlatan tetikleyiciler (örneğin, bir dalda yapılan değişiklikler, pull request'ler).

**Actions** : Bir iş içerisinde yapılan adımlardır. Örneğin, bir test çalışma, derleme ya da bağımlılıkları yükleme gibi adımlar olabilir. Hazır olarak GitHub Marketplace'te yüzlerce action mevcuttur.

**Workflow**: Git deposundaki **.github/workflows** dizininde tanımlanan YAML dosyaları ile belirtilir. Bu dosya, hangi olayların iş akışını tetikleyeceğini ve hangi adımların (actions) çalıştırılacağını belirtir..

**Job**: Bir iş akışı içinde birden fazla görev seti. Paralel veya sırayla çalışabilir.

**Runners** : İş akışındaki eylemleri çalıştmak için kullanılan ortamlar. Örneğin, ubuntu-latest, windows-latest, macOS-latest.

# GitHub Actions Avantajları

**Tam Entegrasyon:** GitHub Actions, GitHub reposuna doğrudan entegre çalışır. Kod değişiklikleri, pull request'ler gibi işlemlerle otomatik tetiklenir. Bu sayede dış araçlara ihtiyaç duymazsınız.

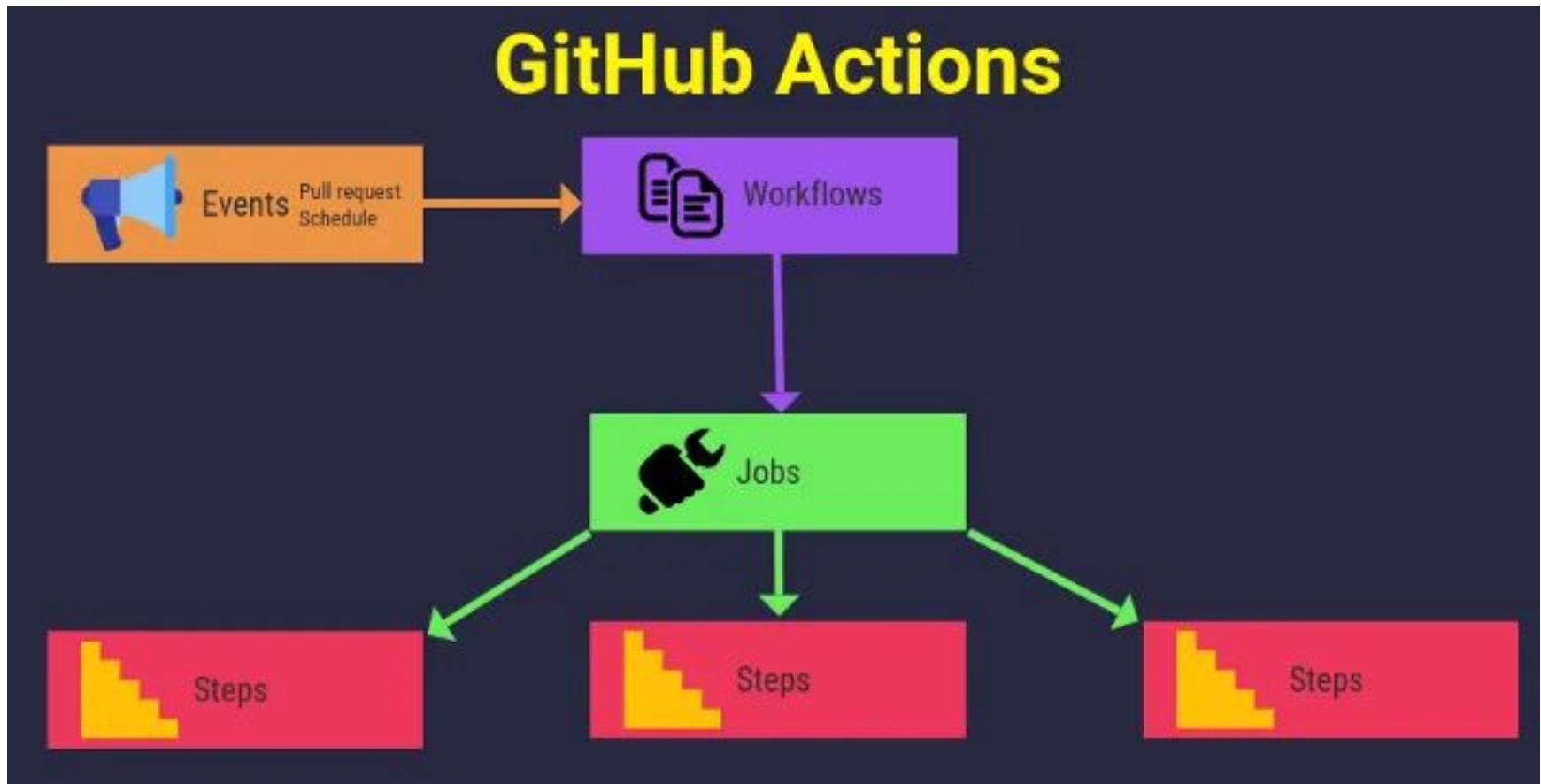
**Özelleştirilebilir İş Akışları:** İş akışları tamamen özelleştirilebilir. Farklı tetikleyiciler ve koşullar ile YAML dosyaları üzerinden projenize özel süreçler oluşturabilirsiniz. ; farklı projeler, diller veya platformlar için özel iş akışları oluşturabilirsiniz.

**Topluluk Desteği:** GitHub Marketplace üzerinden binlerce hazır aksiyon ve şablon erişebilirsiniz. Topluluk tarafından sürekli geliştirilen aksiyonlarla iş akışlarınıza hızla

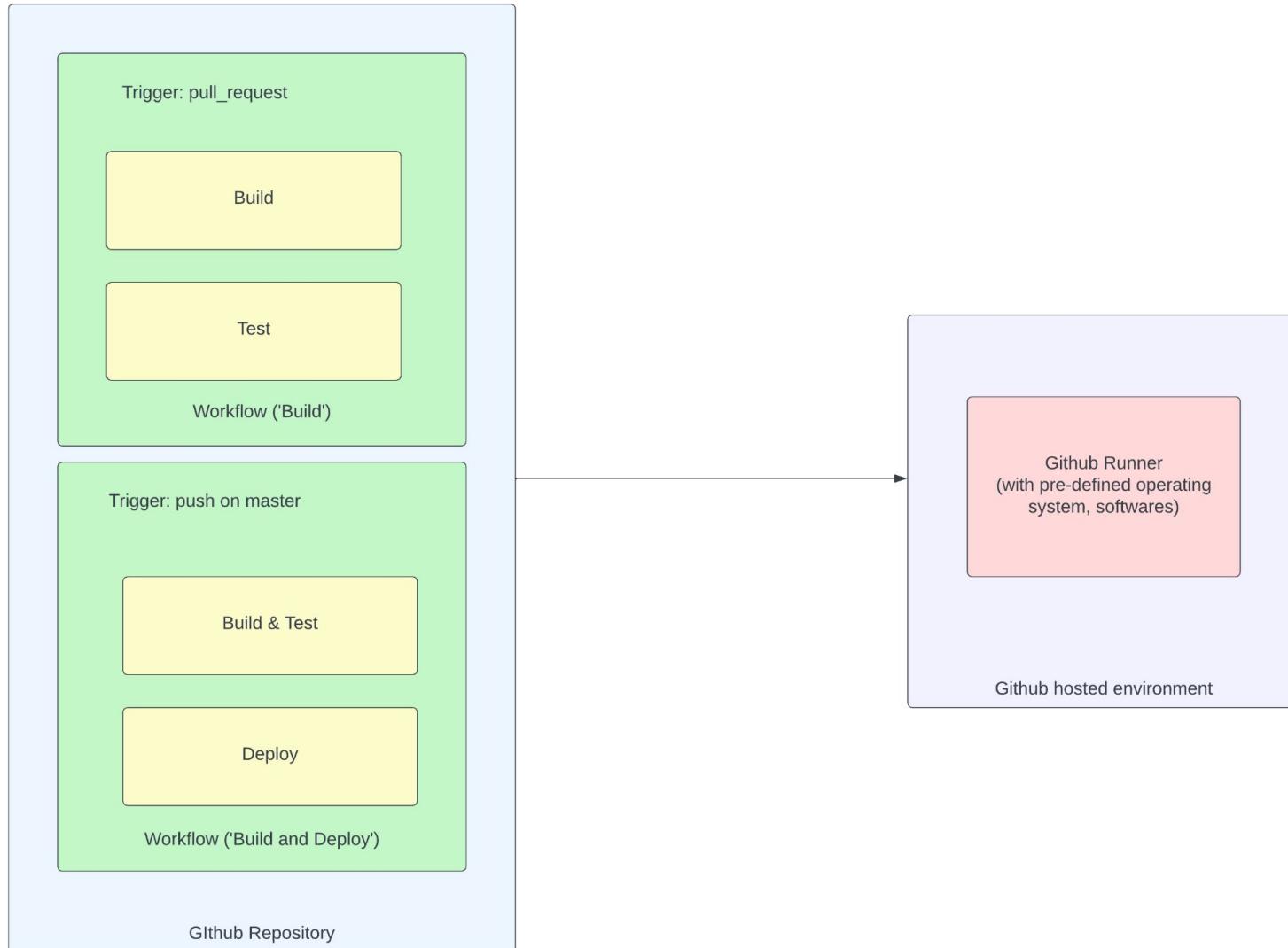
**Sunucu Gerektirmez:** GitHub'ın sunduğu sunucularda çalışır, altyapı yönetimi gerekmez. Aynı zamanda dileyenler için **self-hosted** (kendi barındırılan) runner desteği de mevcuttur.

**Dağıtım Otomasyonu:** Uygulamaları sunuculara veya bulut platformlarına otomatik olarak dağıtabilirsiniz.

# GitHub Actions Nasıl Çalışıyor



# GitHub Actions Nasıl Çalışıyor



# GitHub Actions Workflow Yapısı

```
name: Build and Test

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '14'

      - name: Install dependencies
        run: npm install

      - name: Run tests
        run: npm test
```

## Açıklama:

- **on**: İş akışını tetikleyen olay. Bu örnekte **main** dalına yapılan her push işlemi.
- **jobs**: Bir veya daha fazla iş tanımlar.
- **runs-on**: Hangi ortamda çalıştırılacağını belirler (**ubuntu-latest**).
- **steps**: Her adım, bir komut veya bir GitHub Action kullanır.

# GitHub Actions Market Place

GitHub Action topluluğunda birçok hazır aksiyon bulunur. Bu aksiyonlar işleri hızlandırır.

<https://github.com/marketplace?type=actions>

- **actions/checkout**: Kodu kontrol eder.
- **actions/setup-node**: Node.js ortamını kurar.
- **peaceiris/actions-gh-pages**: Projeyi GitHub Pages'e dağıtır.

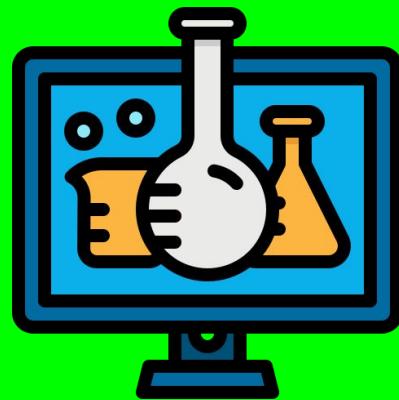
The screenshot shows the GitHub Marketplace interface. At the top, there's a search bar with placeholder text 'Type ⌘ to search'. Below it, a banner says 'Enhance your workflow with extensions' and 'Tools from the community and partners to simplify tasks and automate processes'. On the left, a sidebar has categories like 'Featured', 'Copilot', 'Models', 'Apps', 'Actions', and 'All actions' (which is selected). The main area is titled 'Actions' with the subtitle 'Automate your workflow from idea to production'. It includes filters for 'Filter: All', 'By: All creators', and 'Sort: Popularity'. There are several action cards: 'TruffleHog OSS' (Scan Github Actions with TruffleHog), 'Metrics embed' (An infographics generator with 40+ plugins and 300+ options to display stats about your GitHub account), 'yq - portable yaml processor' (create, read, update, delete, merge, validate and do more with yaml), 'Super-Linter' (Super-linter is a ready-to-run collection of linters and code analyzers, to help validate your source code), 'Gosec Security Checker' (Runs the gosec security checker), and 'OpenCommit — improve commits with ...' (Replaces lame commit messages with meaningful AI-generated messages when you push to remote).

# GitHub Actions Price

<https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions>

Plan	Storage	Minutes (per month)
GitHub Free	500 MB	2,000
GitHub Pro	1 GB	3,000
GitHub Free for organizations	500 MB	2,000
GitHub Team	2 GB	3,000
GitHub Enterprise Cloud	50 GB	50,000

# Github Actions Labs



# ArgoCD



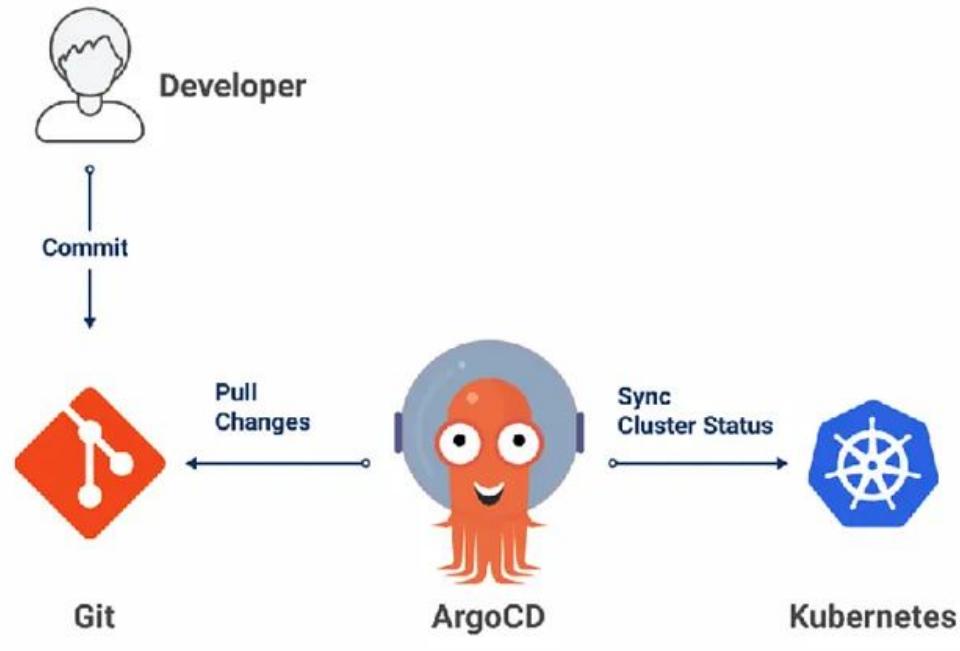
# ArgoCD

## ArgoCD Nedir?

- ArgoCD, Kubernetes için bir GitOps sürekli dağıtım (CD) aracıdır.
- Kullanıcıların uygulamalarını Git depo kaynakları ile senkronize bir şekilde yönetmelerine olanak tanır.

## Neden ArgoCD?

- **Sürüm Kontrolü:** Tüm Kubernetes kaynakları bir Git deposunda sürümlenir.
- **Otomasyon:** Değişiklikler otomatik olarak uygulanır, manuel hatalar azaltılır.
- **Şeffaflık:** Uygulama durumu her zaman izlenebilir.



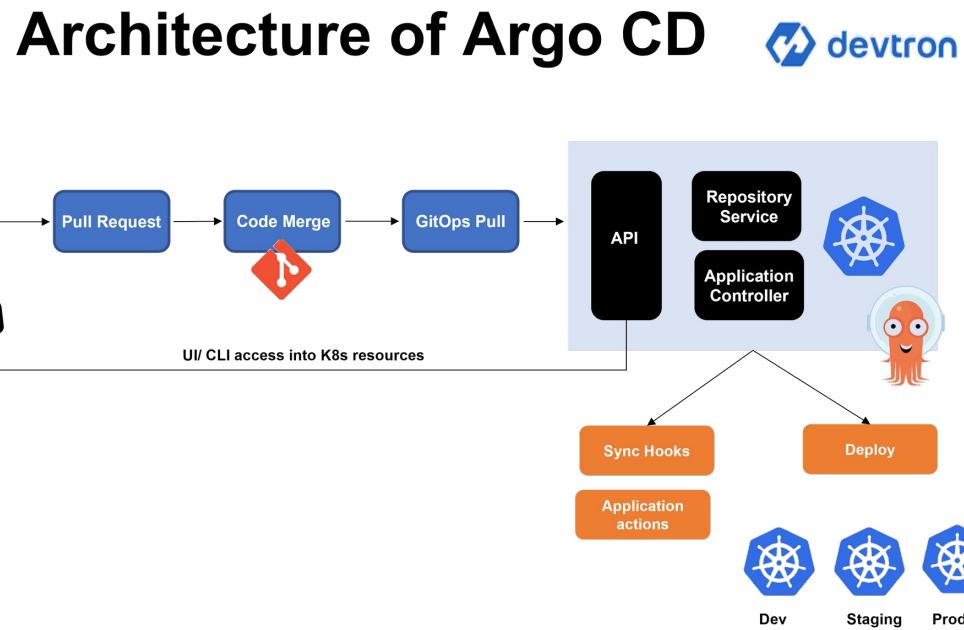
# ArgoCD Mimarisi

## Bileşenler

- **API Server:** Kullanıcı ve sistem bileşenleri arasındaki iletişimini sağlar.
- **Controller:** Hedef durumu uygulamak için sürekli izleme yapar.
- **Repo Server:** Git depolarından kaynakları getirir ve işler.
- **Application Controller:** Uygulama durumunu kontrol eder ve senkronizasyon sağlar.

## Application Project

- ArgoCD'de Application Project, bir veya daha fazla uygulamayı yöneten bir yapılandırılmıştır.
- **Amaç:** Uygulama projeleri, erişim kontrolü ve kaynak yönetimi sağlamak için kullanılır. Örneğin, belirli bir uygulama grubunu belirli bir namespace veya kaynak kısıtlaması altında yönetmek.



# ArgoCD Kurulum

## Kurulum

- **Gereksinimler**

1. **Kubernetes Cluster:** Minikube, EKS, GKE veya başka bir Kubernetes dağıtıımı.
2. **kubectl Aracı:** Kubernetes ile etkileşim kurmak için.
3. **Git Servisi:** GitHub, GitLab, Bitbucket vb.

## ArgoCD'yi Kurma

```
kubectl create namespace argocd
```

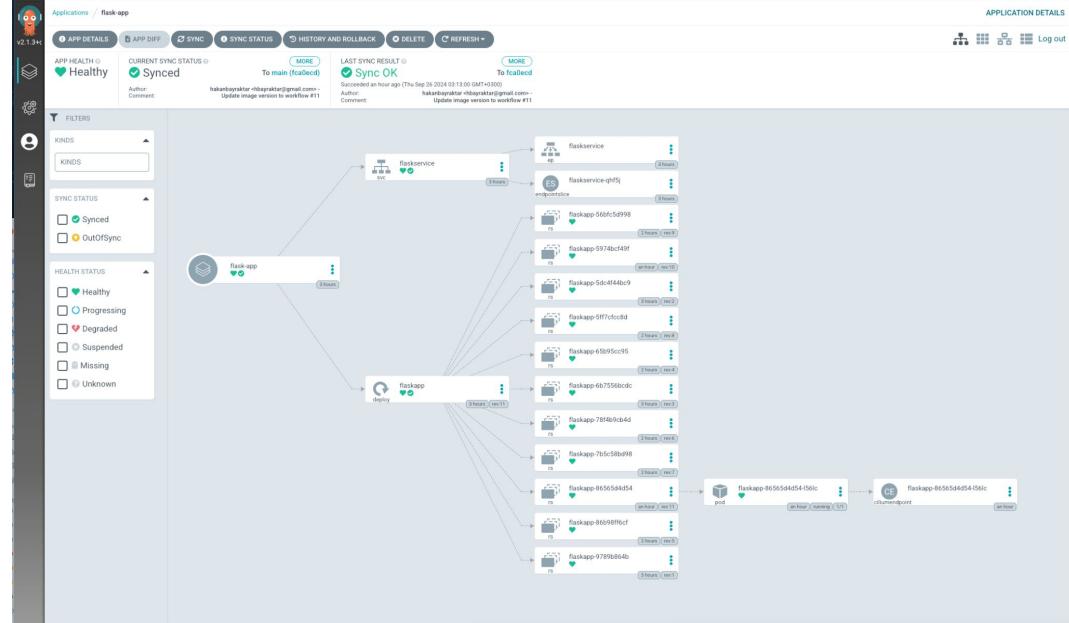
```
kubectl apply -n argocd -f
```

```
https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

## ArgoCD API Server'a Erişim

- LoadBalancer veya NodePort ile dışa açın.
- Erişim adresini öğrenmek için:

```
kubectl get svc -n argocd
```



# Jenkins



# CI/CD Tools

## Open Source



Jenkins



GoCD



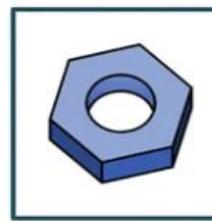
GitLab CI



Drone CI



Spinnaker



Buildbot

## SaaS



Codeship



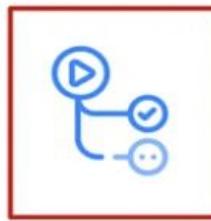
Travis CI



TeamCity



CircleCI



GitHub Actions



Semaphore

## Cloud Services



Azure DevOps



AWS CodePipeline



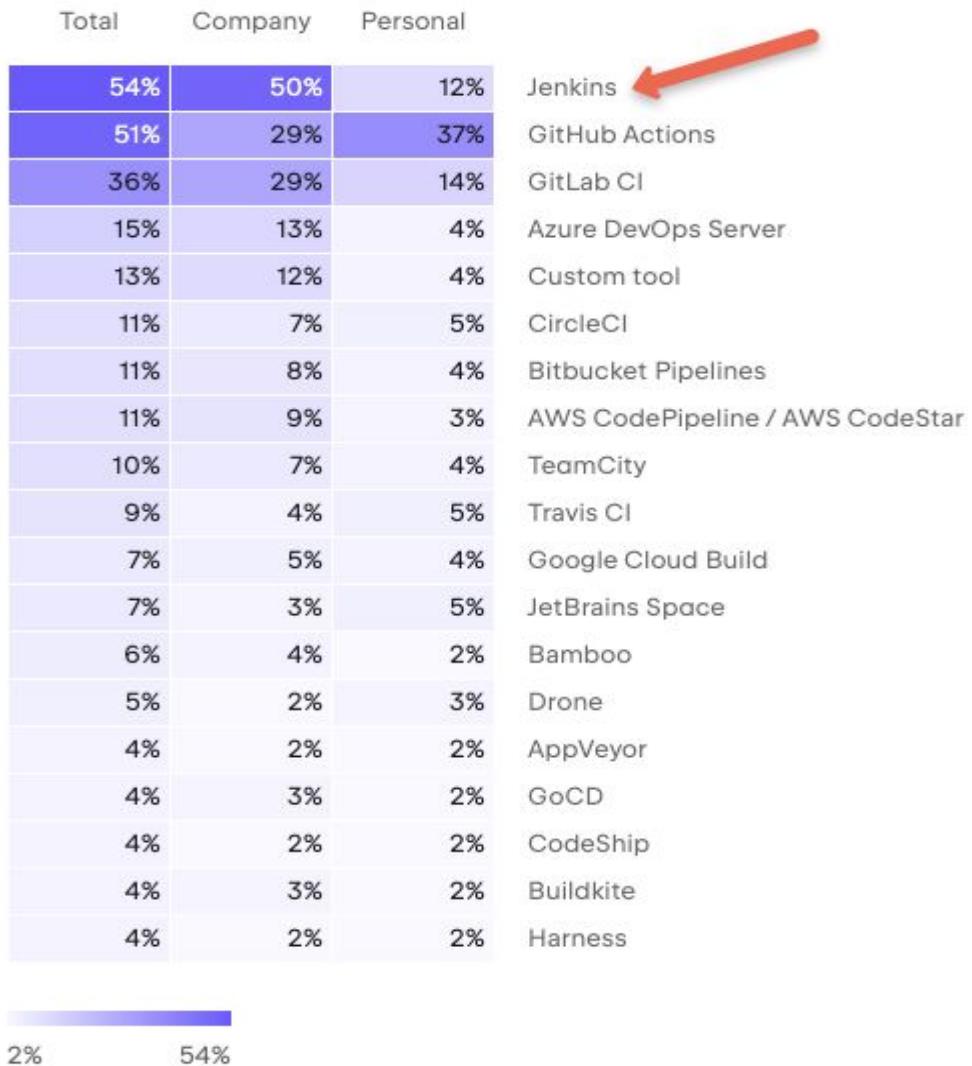
Google Cloud Build

# Top 5 CI/CD tools in 2024

	Jenkins	TeamCity	circleci	Bamboo	GitLab
Open source	Yes	No	No	No	No
Ease of use & setup	Medium	Medium	Medium	Medium	Medium
Built-in features	3/5	4/5	4/5	4/5	4/5
Integration	★★★★★	★★★	★★★★★	★★★	★★★★★
Hosting	On premise & Cloud	On premise & Cloud	On premise	On premise & Bitbucket as Cloud	On premise & Cloud
Free version	Yes	Yes	Yes	Yes	Yes
Build agent license pricing	Free	From \$59 per month	From \$15 per month	From \$10 one-off payment	From \$19 per month per user
Supported OSs	Windows, Linux, macOS, Unix-like OS	Linux or MacoS	Windows, Linux, macOS, Solaris, FreeBSD and more	Windows, Linux, macOS, Solaris	Linux distributions: Ubuntu, Debian, CentOS, Oracle Linux

Developer Ecosystem Report, Developerların %54'ü CI/CD için Jenkins kullanmaktadır.

- Jenkins'i daha iyi anlamak için Sürekli Entegrasyon ve Sürekli Dağıtım kavramlarını anlamak önemlidir.
- Jenkins, büyük bir topluluk desteği sahiptir ve hayatınızı kolaylaştıracak birçok açık kaynak ve kurumsal araç ile entegre olabilen birçok eklenti sunmaktadır.
- GitLab CI ve GitHub Actions gibi CI/CD araçlarının popüleritesinin artmasına rağmen, birçok organizasyonda Jenkins geniş bir şekilde kullanılmaya devam etmektedir.



# Jenkins Nedir

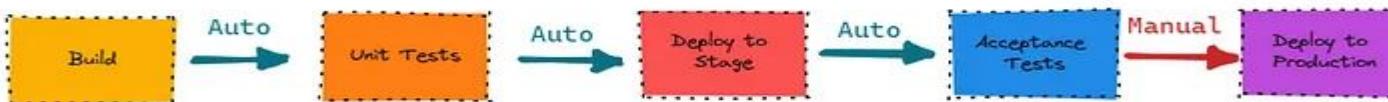
Jenkins, sürekli entegrasyon (CI) ve sürekli dağıtım (CD) süreçlerini yönetmek ve otomatikleştirmek için kullanılan bir açık kaynaklı bir otomasyon aracıdır. Jenkins, yazılım geliştirme süreçlerini daha verimli hale getirerek, yazılım projelerinin sürekli olarak test edilmesi, derlenmesi ve dağıtılmasını sağlar.

**Java programlama dilinde yazılmış, açık kaynaklı bir sürekli entegrasyon /sürekli teslim ve dağıtım (CI/CD) otomasyon yazılımı DevOps aracıdır**

## Continuous Integration



## Continuous Delivery



## Continuous Deployment



# Neden Jenkins?

- **Esneklik:**

- Jenkins, farklı yazılım geliştirme araçları ve süreçleri ile entegre olabilme yeteneğine sahiptir. Farklı programlama dilleri ve framework'leri ile uyumlu çalışabilir.
- Pipeline'lar sayesinde karmaşık süreçleri kodlayarak özelleştirmek mümkündür. Bu, geliştiricilere ve ekip liderlerine kendi iş akışlarını oluşturma imkanı tanır.

- **Geniş Eklenti Desteği:**

- Jenkins, çok sayıda eklenti ile genişletilebilir. Bu eklentiler, versiyon kontrol sistemleri (Git, SVN), test araçları (JUnit, Selenium) ve bulut hizmetleri (AWS, Azure) gibi pek çok aracı destekler.
- Jenkins eklentileri, kullanıcıların ihtiyaçlarına göre iş akışlarını ve otomasyon süreçlerini kişiselleştirmelerine olanak tanır.

- **Topluluk Desteği:**

- Jenkins, büyük ve aktif bir kullanıcı topluluğuna sahiptir. Bu, kullanıcıların sorunlarına hızlı çözümler bulmasını sağlar ve sürekli gelişim için bir kaynak oluşturur.
- Forumlar, belgeler ve çevrimiçi kaynaklar sayesinde kullanıcılar, Jenkins ile ilgili en güncel bilgilere ve en iyi uygulamalara ulaşabilirler.

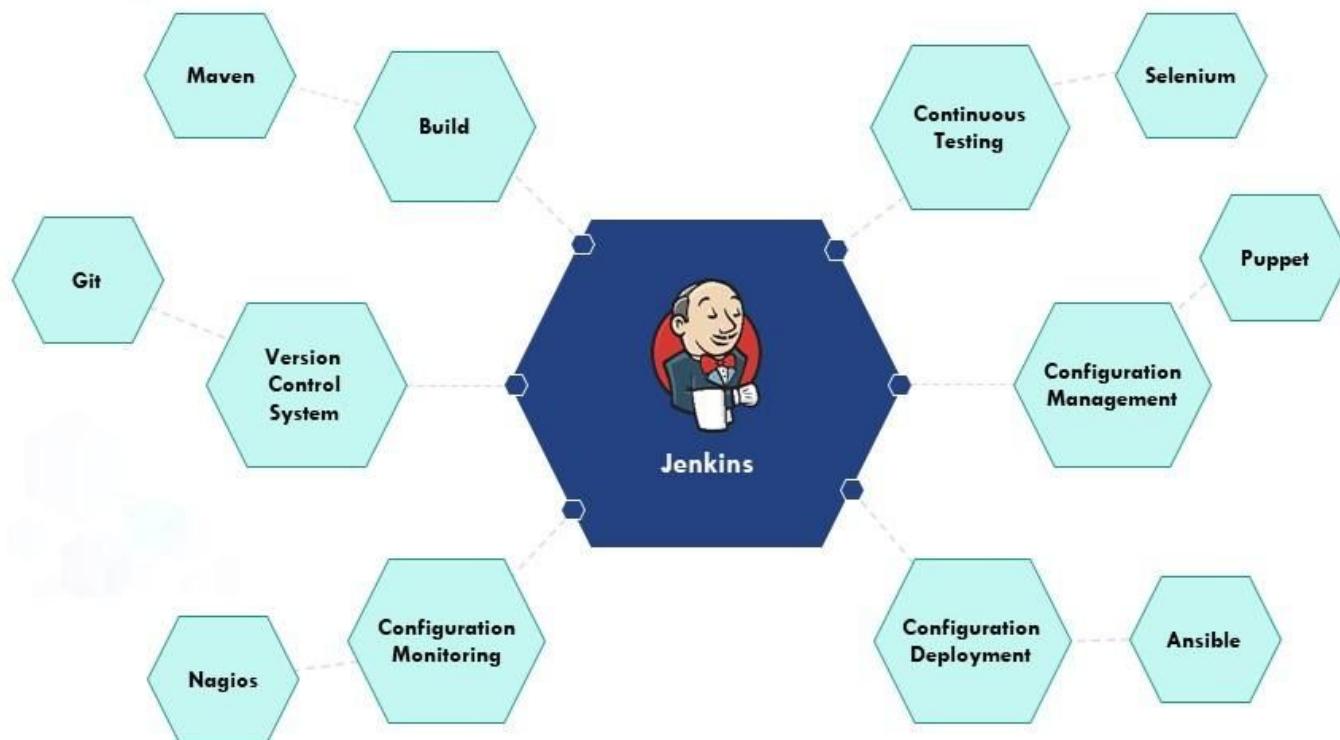
# Jenkins Hangi Amaçlarla Kullanılır?

- Uygulama ve altyapı kodu için Sürekli Entegrasyon.
- Farklı ortamlara uygulama dağıtmak için Jenkins pipeline olarak kod ile Sürekli Dağıtım boru hatları.
- Altyapı bileşenlerinin dağıtımı ve yönetimi için IaC (Infrastructure as Code) araçları.
- Jenkins işleri kullanarak toplu işlemler çalışma.
- Yedekleme, temizleme, uzaktan script çalışma, olay tetikleme gibi ad-hoc işlemler gerçekleştirmeye.

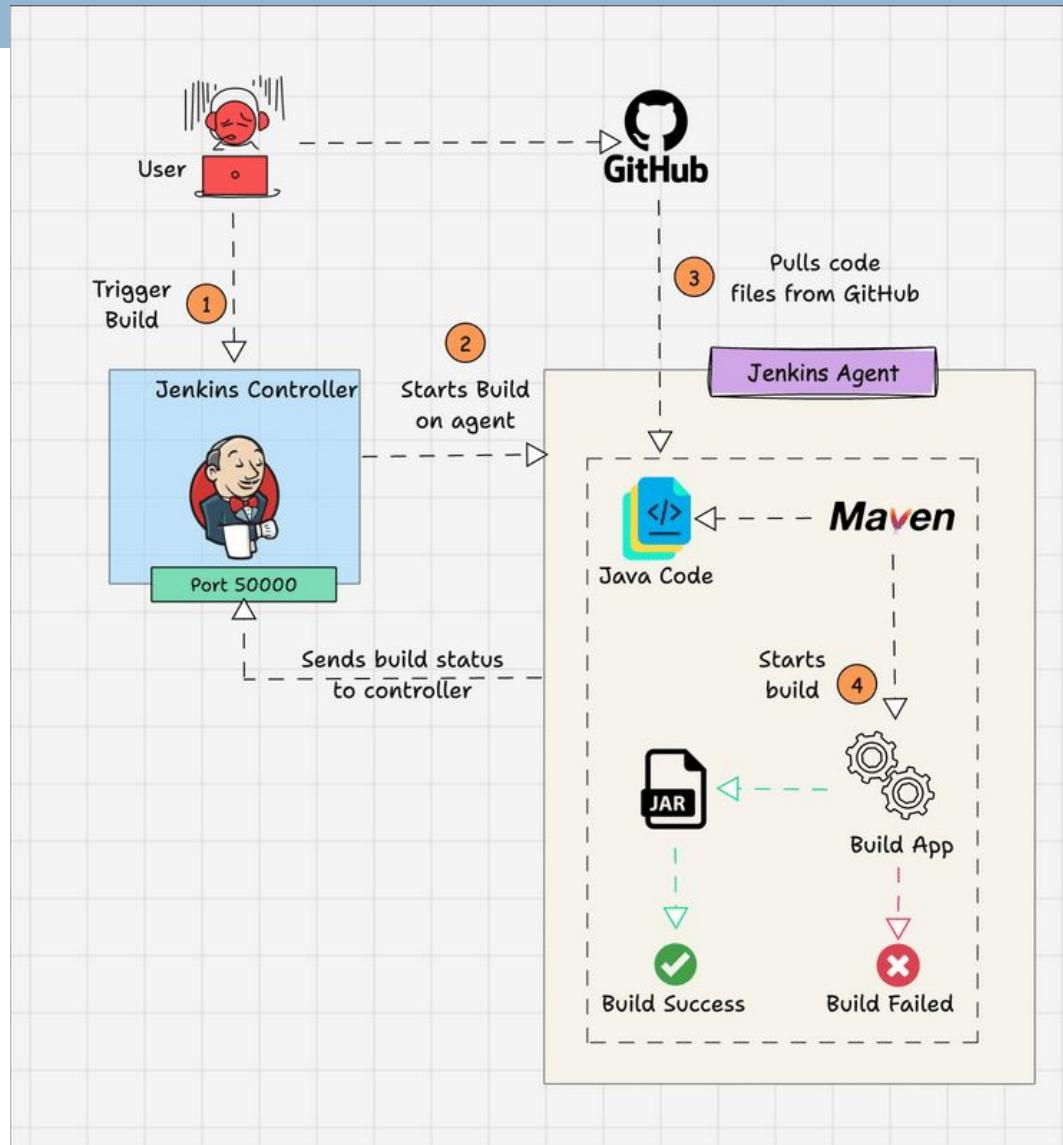
# Features of Jenkins



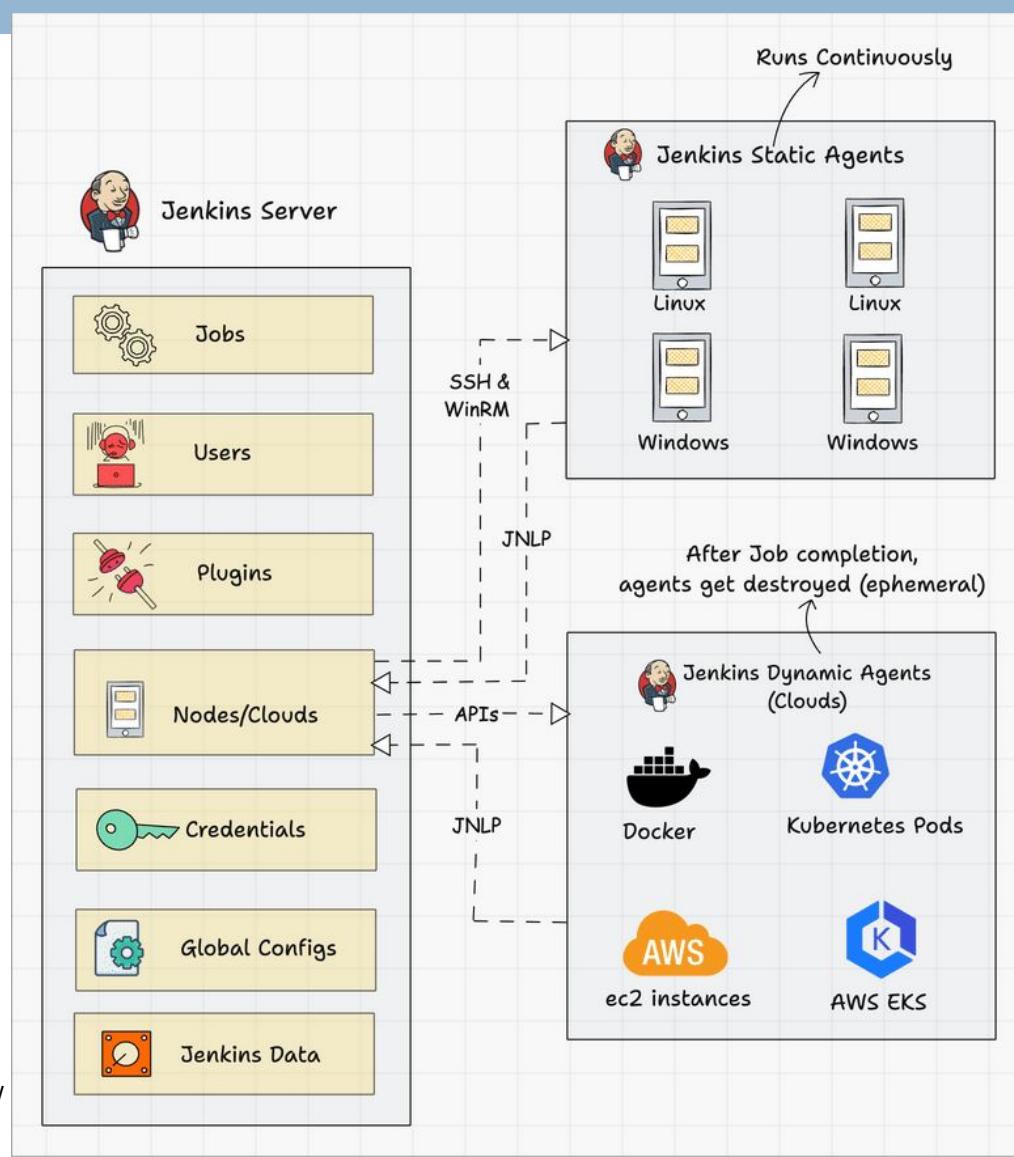
## Continuous Integration in Jenkins



# Jenkins CI



# Jenkins Architecture



# Jenkins Temel Kavramlar

**Job:** Jenkins'te bir iş, belirli bir görevi gerçekleştiren birimdir. İşler, yazılım projelerinin derlenmesi, test edilmesi, dağıtılması gibi adımları içerebilir.

**Build :** Bir işin bir parçası olan işlemlerin bir adımıdır. Derleme adımı, kaynak kodunun derlenmesi, bir uygulamanın oluşturulması veya bir yapı paketinin hazırlanması gibi görevleri içerebilir.

**Pipeline:** Jenkins'te, bir işin ardışık adımlarını tanımlayan ve yöneten kod parçacıklarıdır. İki türü vardır:

- **Declarative Pipeline:** Daha basit ve okunabilir bir yapı sunar.
- **Scripted Pipeline:** Daha esnek ve güçlüdür, Groovy tabanlıdır.

Pipeline'lar, iş akışını, adımları ve koşulları belirlemek için kullanılır ve genellikle **Jenkinsfile** adı verilen bir dosyada tanımlanır.

# Jenkins Temel Kavramlar

**Node:** Jenkins'te çalışan fiziksel veya sanal bir makinedir. Node'lar, işlerin gerçekleştirildiği ve Jenkins'in kaynakları kullandığı yerlerdir.

**Agent :** Jenkins'te bir işin çalıştırılması için kullanılan bir düğümün parçasıdır. Agent'lar, belirli işlerin belirli düğümlerde yürütülmesini sağlar.

**Trigger :** Jenkins'te işlerin ne zaman başlatılacağını belirleyen mekanizmalardır. Zamanlanmış tetikleyiciler, SCM (sürüm kontrolü) değişiklikleri, başka bir işin tamamlanması gibi durumlar tetikleyici olabilir.

**Plugin :** Jenkins işlevsellliğini genişletmek için kullanılan yazılım parçalarıdır. Eklentiler, Jenkins'e yeni özellikler eklemek veya mevcut özellikleri genişletmek için kullanılabilir.

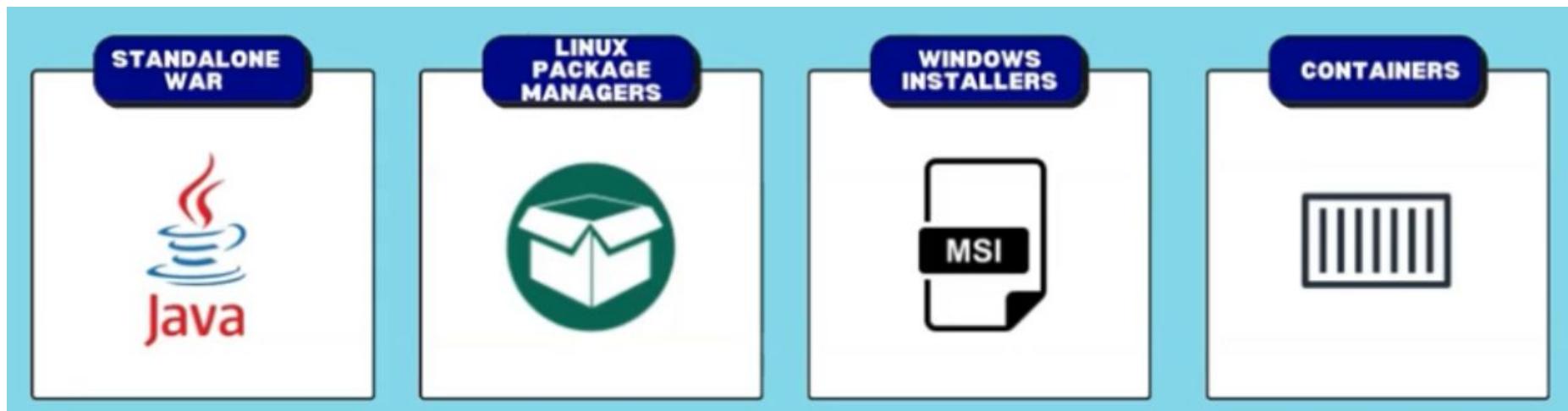
# Jenkins Temel Kavramlar

**Workspace (Çalışma Alanı):** Her bir iş için Jenkins tarafından oluşturulan geçici bir dizin. Derleme ve test dosyaları burada tutulur.

**Artifact (Nesne):** Derleme işlemi sonucunda üretilen dosyalar. Yazılım dağıtıımı için gereken dosyalar burada bulunur.

**View (Görünüm):** Jenkins'teki işlerin veya projelerin gruplandığı ve görüntüülendiği bir arayüz. Farklı kullanıcılar için özelleştirilebilir.

# Jenkins installation channels



# PREREQUISITES

## JENKINS RELEASES

### LTS

- Released every twelve weeks

### WEEKLY

- Released weekly to deliver bug fixes and features

## PREREQUISITES

- 256 MB RAM
- 1 GB of disk space [10 GB if running as a Docker container]
- Java 11 or 17

- Modern web browsers – Google Chrome, Apple Safari, Mozilla Firefox, Microsoft Edge
- Port 8080

## JENKINS INSTALLATION CHANNELS

### STANDALONE WAR



### LINUX PACKAGE MANAGERS



### WINDOWS INSTALLERS



### CONTAINERS



# JENKINS Installation

## INSTALLATION STEPS

1

### IMPORT THE GPG KEY FOR JENKINS REPOSITORY

```
curl -fsSL  
https://pkg.jenkins.io/debian-  
stable/jenkins.io-2023.key | sudo  
tee \  
/usr/share/keyrings/jenkins-  
keyring.asc > /dev/null
```

2

### ADD THE REPOSITORY TO THE LIST OF SOURCES

```
echo deb [signed-  
by=/usr/share/keyrings/jenkins-  
keyring.asc] \  
https://pkg.jenkins.io/debian-  
stable binary/ | sudo tee \  
/etc/apt/sources.list.d/jenkins.list
```

3

### UPDATE PACKAGE INDEX

- sudo apt-get update

4

### INSTALL JENKINS

```
sudo apt-get install jenkins
```

# JENKINS Dashboard

The screenshot shows the Jenkins dashboard interface with several labeled components:

- Header:** Located at the top, featuring the Jenkins logo, a search bar ("Search (⌘+K)"), and user authentication links ("Butler" and "Log out").
- Breadcrumbs:** A navigation path starting from "Dashboard".
- Side Navigation:** A sidebar on the left containing links for "New Item", "People", "Build History", "Manage Jenkins", and "My Views".
- All View:** A button labeled "All" with a plus sign, indicating a view for all jobs.
- Add description:** A link to add a description for the current view.
- Monitoring Builds:** A section showing the status of four build jobs: "backend-app", "custom-app", "frontend-app", and "pipeline-app". Each job has columns for Status (S), Warning (W), Name, Last Success, Last Failure, and Last Duration.
- All jobs:** A label pointing to the list of four build jobs.
- Build Queue:** A section stating "No builds in the queue."
- Build Executor Status:** A section showing "1 Idle" and "2 frontend-app" (one of which is highlighted).
- Icon legend:** A legend for build status icons: S (green checkmark), M (yellow sun), L (red cloud).
- Atom feed links:** Links for "Atom feed for all", "Atom feed for failures", and "Atom feed for just latest builds".
- Footer:** Located at the bottom of the page, containing links for "REST API" and "Jenkins 2.375.3".

S	W	Name	Last Success	Last Failure	Last Duration
✓	☀️	backend-app	41 min #3	N/A	11 ms
✗	☀️	custom-app	42 min #1	N/A	1 min 59 sec
✓	☀️	frontend-app	41 min #1	N/A	20 sec
✗	☁️	pipeline-app	40 min #1	40 min #2	0.98 sec

# Pluginler(eklentiler) ve Entegrasyonlar

## Pluginlerin Önemi

- **Genişletilebilirlik:** Jenkins, eklentiler aracılığıyla işlevselliğini artırabilir. Kullanıcılar, ihtiyaçlarına uygun eklentileri yükleyerek Jenkins'i özelleştirebilirler.
- **Entegrasyon:** Farklı araçlarla entegrasyon sağlayarak, iş akışlarını daha verimli hale getirir. Örneğin, kaynak kodu yönetim sistemleri, test araçları ve dağıtım platformlarıyla entegrasyon imkanı sunar.



# Plugin Yükleme Adımları

## Plugin Yükleme Adımları

- 1-Jenkins arayüzünde, Manage Jenkins sekmesine gidin.
- 2-Manage Plugins seçeneğini seçin.
- 3-Available sekmesinden ihtiyaç duyduğunuz pluginleri arayın.
- 4-Seçtiğiniz pluginları işaretleyin ve sayfanın altında bulunan Install without restart butonuna tıklayın.

## Temel Jenkins Pluginları

1. **Pipeline Plugin:** Pipeline'ların temel işlevsellliğini sağlar.
2. **Groovy Plugin:** Groovy tabanlı script'lerin çalıştırılmasına olanak tanır.
3. **Git Plugin:** Git reposundan kaynak kodlarını çekmek için gereklidir.
4. **Credentials Plugin:** Gizli bilgilerin (şifreler, token'lar vb.) yönetimi için kullanılır.
5. **Job DSL Plugin:** Job'ları tanımlamak ve yapılandırmak için DSL desteği sunar.
6. **Parameterized Trigger Plugin:** Diğer job'ları parametre ile tetiklemek için kullanılır.
7. **Blue Ocean Plugin:** Modern ve kullanıcı dostu bir kullanıcı arayüzü sunar.
8. **Slack Notification Plugin (veya başka bildirim pluginları):** şlerin durumunu bildirmek için kullanılır.

# Jenkins Pipeline Nedir?

**Pipeline**, Jenkins'te iş akışlarını tanımlamak için kullanılan bir yapıdır. Pipeline, iş akışını kodlayarak daha esnek ve sürdürülebilir hale getirir. **Groovy** tabanlı bir **DSL (Domain-Specific Language)** kullanarak iş akışlarını oluşturabilirsiniz. Pipeline'lar, iki ana türde gelir:

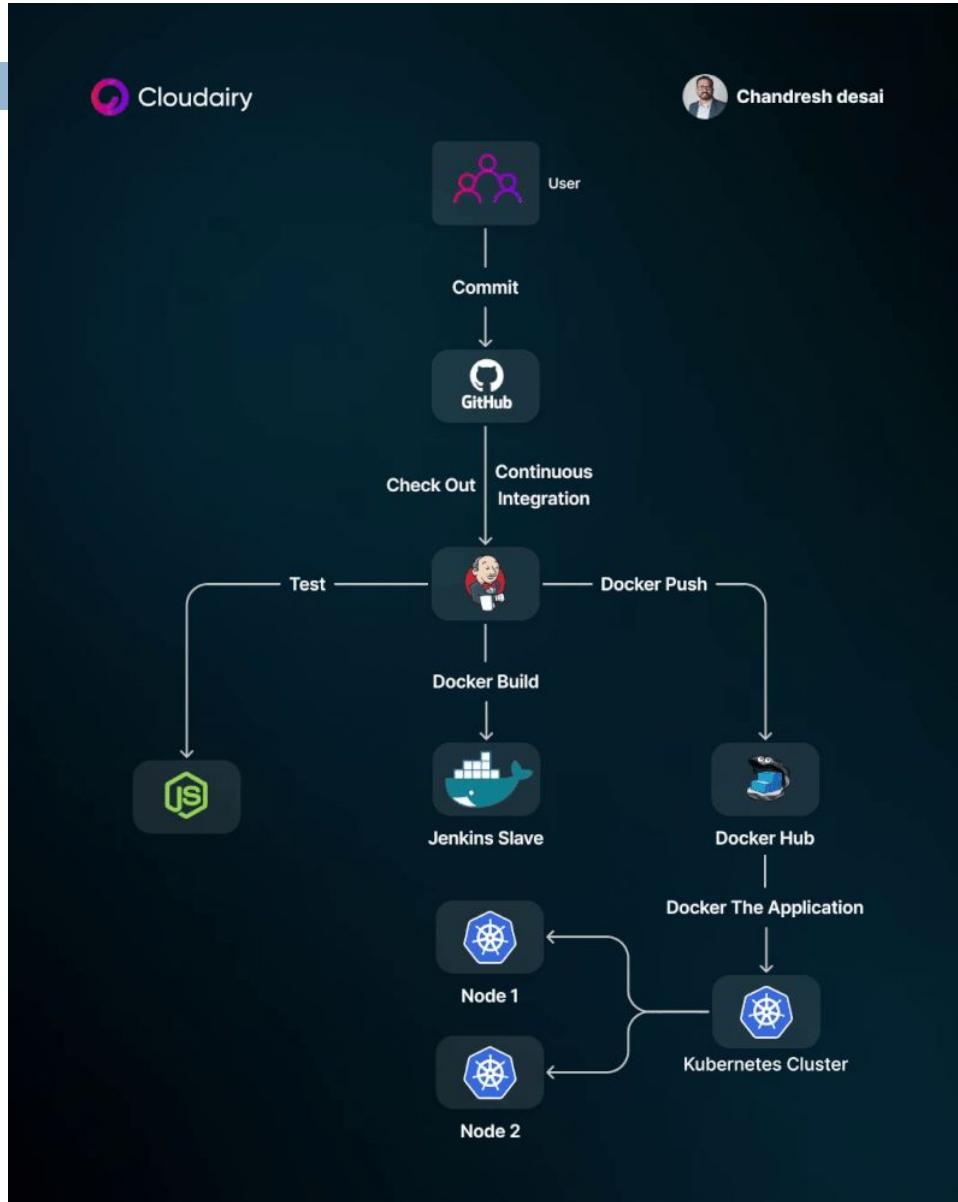
## 1. Declarative Pipeline

Bu tür, daha yapılandırılmış ve okunabilir bir biçim sunar. Kullanıcıların daha az kod yazmasını sağlar ve daha az karmaşıklık içerir.

## 2. Scripted Pipeline

Daha esnek ve karmaşık iş akışları oluşturmanıza olanak tanır. Groovy diline daha fazla hakim olmayı gerektirir.

# Jenkins pipeline



# Pipeline Komutları

**pipeline {}** Bu yapı, bir pipeline'ı tanımlamak için kullanılır. İçinde agent, stages, environment gibi anahtar bileşenleri barındırır.

**agent** Job'ın hangi ortamda çalışacağını belirtir. Örneğin, any ile herhangi bir uygun ajan kullanılabilir.

**stages {}** Pipeline'daki ana adımları tanımlar. Her bir adım bir stage olarak tanımlanır.

**steps {}** Her bir stage içindeki işlemleri tanımlamak için kullanılır.

**script {}** Groovy kodu yazmak için kullanılır. Scripted pipeline'larda sıkça kullanılır.

**environment {}** Pipeline içerisinde kullanılacak ortam değişkenlerini tanımlar.

# Pipeline Komutları

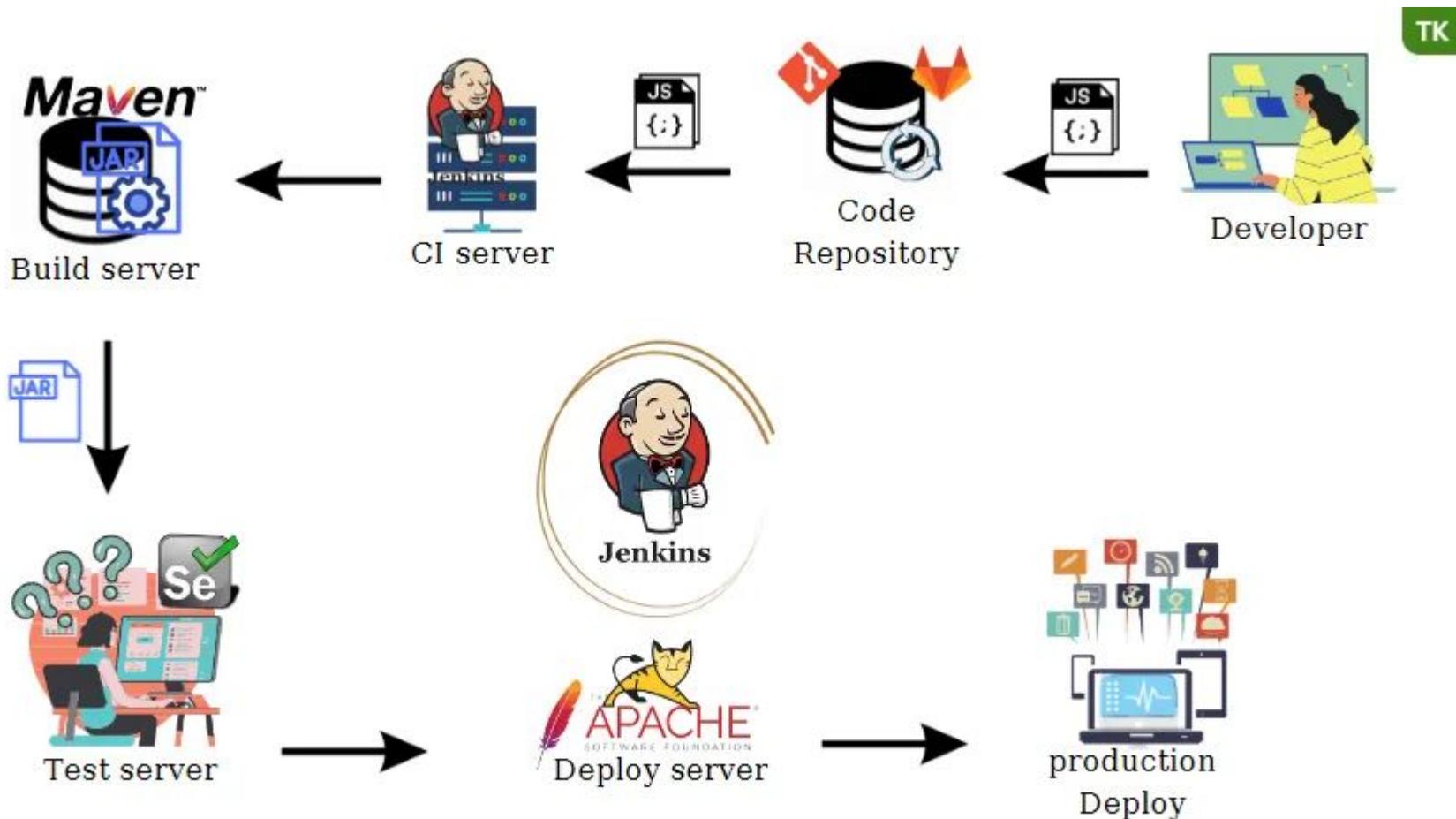
**post {}** Pipeline tamamlandıktan sonra gerçekleştirilecek adımları tanımlar. Örneğin, başarı veya hata durumlarında yapılacaklar.

**parameters {}** Pipeline'a kullanıcıdan girdi almak için kullanılır. Farklı türlerde parametreler tanımlanabilir (örneğin, string, boolean).

**when {}** Belirli koşullara göre adımları veya stage'leri koşullu olarak çalıştırılmak için kullanılır. Örneğin, belirli bir branch'te çalıştırılmak gibi.

**input {}** Pipeline çalışırken kullanıcılarından onay almak için kullanılır. Bu, bir adının belirli bir onay olmadan devam etmemesini sağlar.

# Jenkins pipeline



# Pipeline kurarken aşağıdaki sorulara cevap vermemiz gereklidir

- 1- Kod nerede depolanacak
- 2- Nasıl build edeceğim
- 3- Bu kodu Nasıl ve ne ile test edeceğim
- 5- Jenkins makinam hangi servislerle iletişime geçecek
- 6- Triger (tetikleyici) işlemi nasıl olacak

# Basit Declarative Pipeline

```
pipeline {  
    agent any  
  
    stages {  
        stage('Build') {  
            steps {  
                echo 'Building...'  
                // Derleme komutları  
            }  
        }  
        stage('Test') {  
            steps {  
                echo 'Testing...'  
                // Test komutları  
            }  
        }  
        stage('Deploy') {  
            steps {  
                echo 'Deploying...'  
                // Dağıtım komutları  
            }  
        }  
    }  
}
```

# Scripted Pipeline

```
node {  
    stage('Build') {  
        echo 'Building...'  
        // Derleme komutları  
    }  
    stage('Test') {  
        echo 'Testing...'  
        // Test komutları  
    }  
    stage('Deploy') {  
        echo 'Deploying...'  
        // Dağıtım komutları  
    }  
}
```

# Parametre örneği Pipeline

groovy

 Copy code

```
pipeline {
    agent any
    parameters {
        string(name: 'ENVIRONMENT', defaultValue: 'staging', description: 'Deploy environment')
    }
    stages {
        stage('Build') {
            steps {
                echo "Building for ${params.ENVIRONMENT}..."
            }
        }
        stage('Test') {
            steps {
                echo "Testing for ${params.ENVIRONMENT}..."
            }
        }
        stage('Deploy') {
            steps {
                echo "Deploying to ${params.ENVIRONMENT}..."
            }
        }
    }
}
```

# Koşullu Pipeline

groovy

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Building...'
            }
        }
        stage('Deploy') {
            when {
                branch 'main'
            }
            steps {
                echo 'Deploying to production...'
            }
        }
    }
}
```

# Jenkinsfile (Conditional)

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                // Build işlemleri burada tanımlanır
            }
        }
        stage('Test') {
            steps {
                // Test işlemleri burada tanımlanır
            }
            when {
                branch 'master' // Sadece 'master' dalına sahipse bu adımı yürüt
            }
        }
        stage('Deploy') {
            steps {
                // Dağıtım işlemleri burada tanımlanır
            }
            when {
                environment name: 'PRODUCTION', value: 'true' // Yalnızca 'PRODUCTION' olsun
            }
        }
    }
}
```

# Kullanıcı onayı gerektiren Pipeline

```
pipeline {  
    agent any  
  
    stages {  
        stage('Build') {  
            steps {  
                echo 'Building...'  
            }  
        }  
        stage('Approval') {  
            steps {  
                input 'Onayınızı bekliyoruz'  
            }  
        }  
        stage('Deploy') {  
            steps {  
                echo 'Deploying...'  
            }  
        }  
    }  
}
```

# Jenkinsfile (parallel)

```
pipeline {  
    agent any  
    stages {  
        stage('Build & Test') {  
            steps {  
                parallel(  
                    "Build": {  
                        // Build işlemleri burada tanımlanır  
                    },  
                    "Test": {  
                        // Test işlemleri burada tanımlanır  
                    }  
                )  
            }  
        }  
        stage('Deploy') {  
            steps {  
                // Dağıtım işlemleri burada tanımlanır  
            }  
        }  
    }  
}
```

# Jenkinsfile (trigger)

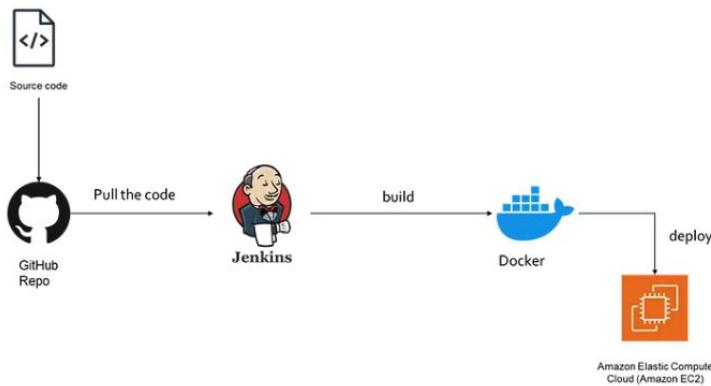
```
pipeline {
    agent any
    triggers {
        scm('*/* * * * *') // SCM sistemi her 5 dakikada bir kontrol edilir
    }
    stages {
        stage('Build') {
            steps {
                // Build adımları burada tanımlanır
            }
        }
        // Diğer aşamalar burada tanımlanır
    }
}
```

# Jenkins Labs



# Jenkins Labs

- Jenkins Kurulum
- Jenkins Pipeline oluşturma
- Jenkins Pluginleri
- Jenkins ve Git Entegrasyonu
- Jenkins ve Maven Entegrasyonu
- Jenkins ve Docker Entegrasyonu
- Jenkins ve Kubernetes Entegrasyonu



# Gitlab CI/CD



# Gitlab CI/CD Nedir?

**GitLab**, kod depolama, proje yönetimi, sürekli entegrasyon/sürekli dağıtım (CI/CD), iş takibi, kod incelemesi, wiki ve daha fazlasını sağlayan bir yazılım paketidir

**GitLab CI/CD**, GitLab platformunun bir parçasıdır ve sürekli entegrasyon (CI) ve sürekli dağıtım (CD) süreçlerini destekler.

Yazılım geliştirme süreçlerinde kodun sürekli olarak test edilmesi, derlenmesi ve dağıtılması için otomatik iş akışları oluşturur.

## Ana Bileşenler

- **Pipeline**: CI/CD süreçlerinin sıralı adımlarını tanımlayan bir yapı. Her pipeline, farklı aşamalardan oluşur.
- **Job**: Pipeline içindeki belirli bir iş. Örneğin, kod derleme veya test etme.
- **Stage**: Bir grup job'u içeren aşama. Örneğin, bir "test" aşaması altında birden fazla test job'u olabilir.

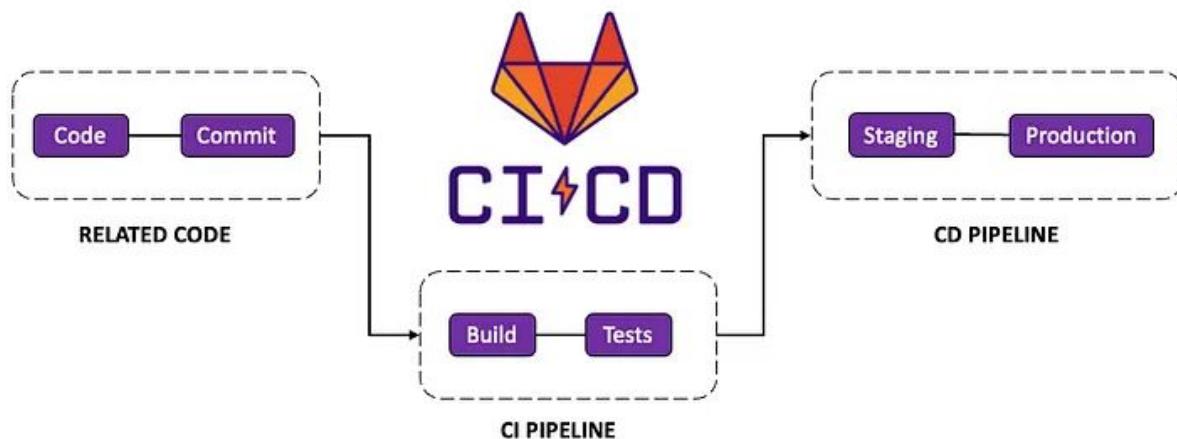
# Gitlab CI/CD Süreci

## Sürekli Entegrasyon (CI):

- Kod değişiklikleri yapıldığında otomatik olarak testlerin çalıştırılması.
- Kod kalitesinin artırılması ve hataların erken tespit edilmesi.

## Sürekli Dağıtım (CD):

- Başarılı testlerin ardından otomatik olarak uygulamanın üretim ortamına dağıtılması.
- Kullanıcıların yeni özelliklere daha hızlı erişebilmesi.



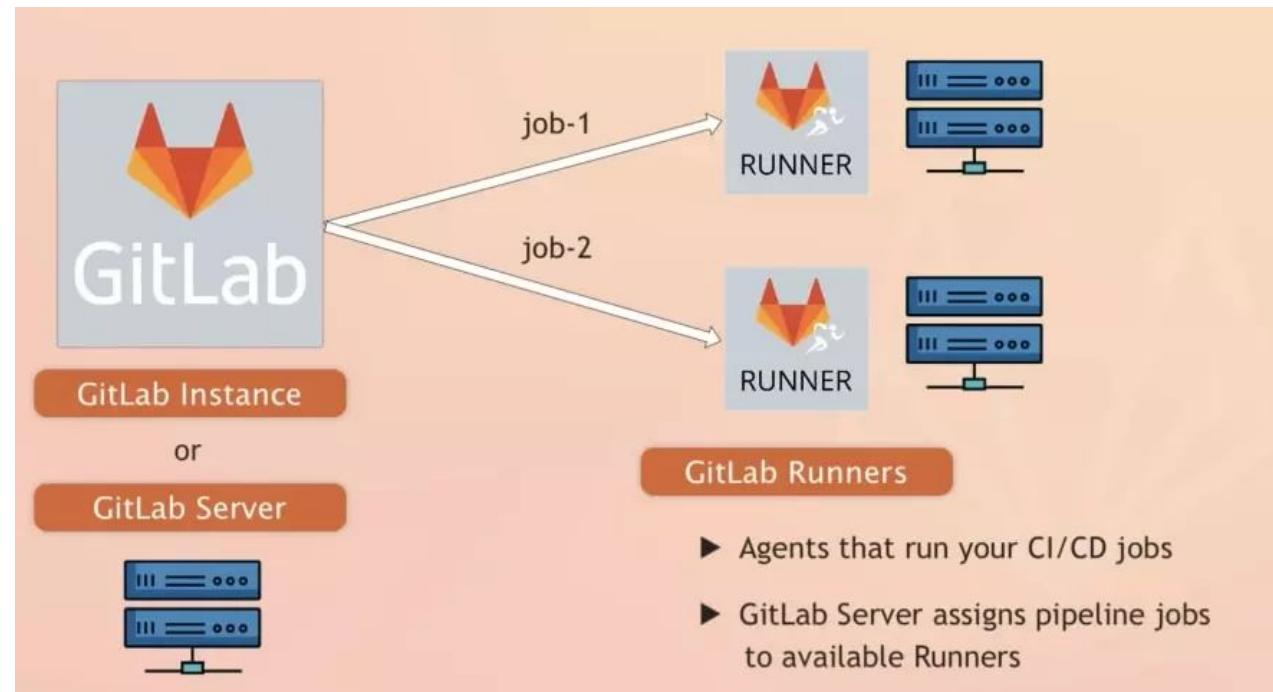
# Gitlab Runner

**GitLab Runner**, GitLab CI/CD süreçlerini gerçekleştiren bir uygulamadır. İş akışlarını (pipeline) çalıştırmak için gerekli olan görevleri yerine getirir.

## Runner Türleri

**Shared Runners:** Tüm projeler tarafından kullanılabilen genel runner'lar. GitLab.com üzerinde varsayılan olarak mevcuttur.

**Specific Runners:** Belirli bir projeye atanmış olan runner'lar. Bu runner'lar sadece o projeye ait job'ları çalıştırır.



# Gitlab Workflow

**.gitlab-ci.yml Dosyası:** Pipeline'ın yapılandırıldığı dosya. Burada job'lar, aşamalar ve diğer parametreler tanımlanır.

Örnek bir .gitlab-ci.yml dosyası

```
stages:
  - build
  - test
  - deploy

workflow:
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'

build_job:
  stage: build
  script:
    - echo "Building the project..."

test_job:
  stage: test
  script:
    - echo "Running tests..."

deploy_job:
  stage: deploy
  script:
    - echo "Deploying the application..."
only:
  - main
```

# Gitlab Workflow komutları

**workflow:** Bir pipeline'ın hangi koşullar altında çalışacağını belirler. Örneğin, belirli bir branch'te veya belirli bir değişiklik olduğunda çalışacak şekilde ayarlanabilir.

**rules:** job'ların veya pipeline'ların çalışıp çalışmayacağını belirlemek için kullanılabilir. Birden fazla koşul tanımlanabilir.

**only:** Belirli branch'lerde veya tag'lerde job'ların çalışmasını sağlar.

**except:** Belirli branch'lerde veya tag'lerde job'ların çalışmasını engeller.

**when :** Job'ın ne zaman çalışacağını belirtir. Örneğin, bir job'ın belirli bir koşul sağlandığında hemen çalışmasını isteyebilirsiniz.

# Monitoring & Logging



# Devops'ta Monitoring Önemi

- Anında Sorun Tespiti
- Performans takibi
- Güvenlik Tehditlerini görme
- Kullanıcıların yazılımı nasıl kullandığını anlamak için geri bildirim sağlar.
- Uygulama ve altyapının genel sağlığını izleyerek potansiyel sorunları önceden tespit etmeye yardımcı olur.
- Sorunlar meydana gelmeden önce önleyici bakım yapılmasını sağlar.
- Sistem kaynaklarının daha verimli kullanılmasını sağlar ve israfı önler.
- Monitoring Hizmet Seviyesi Anlaşmalarının (SLA) karşılanması sağlanmak için performansı ve kullanılabilirliği kontrol eder.
- Olaylar meydana geldiğinde hızlı müdahale ve çözüm sağlar.
- İzleme, süreçleri ve performansı sürekli iyileştirmek için geri bildirim sunar.



# DevOps'ta Loglama (Kayıt Tutmanın) Önemi

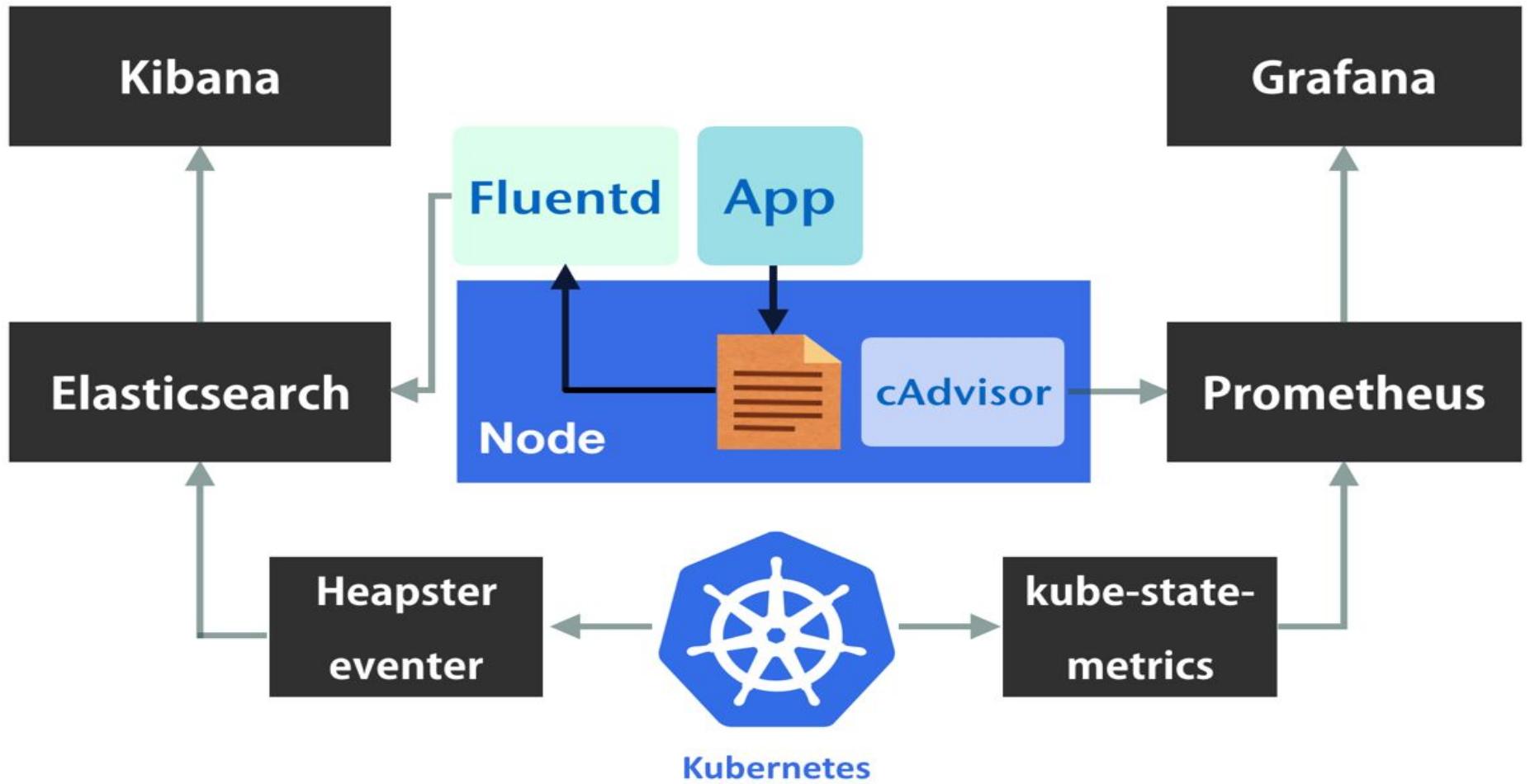
- Güvenlik İzlemesi
- Uyumluluk
- Proaktif Bakım
- Detaylı Analiz
- Takım İletişimi ve Hata Teşhisи
- Performans İzleme
- Kullanıcı Aktivitesi Takibi
- Hata Ayıklama (Troubleshooting)
- Desen(Pattern) ve Trend Tanımlama



# DevOps Monitoring & Logging Tools

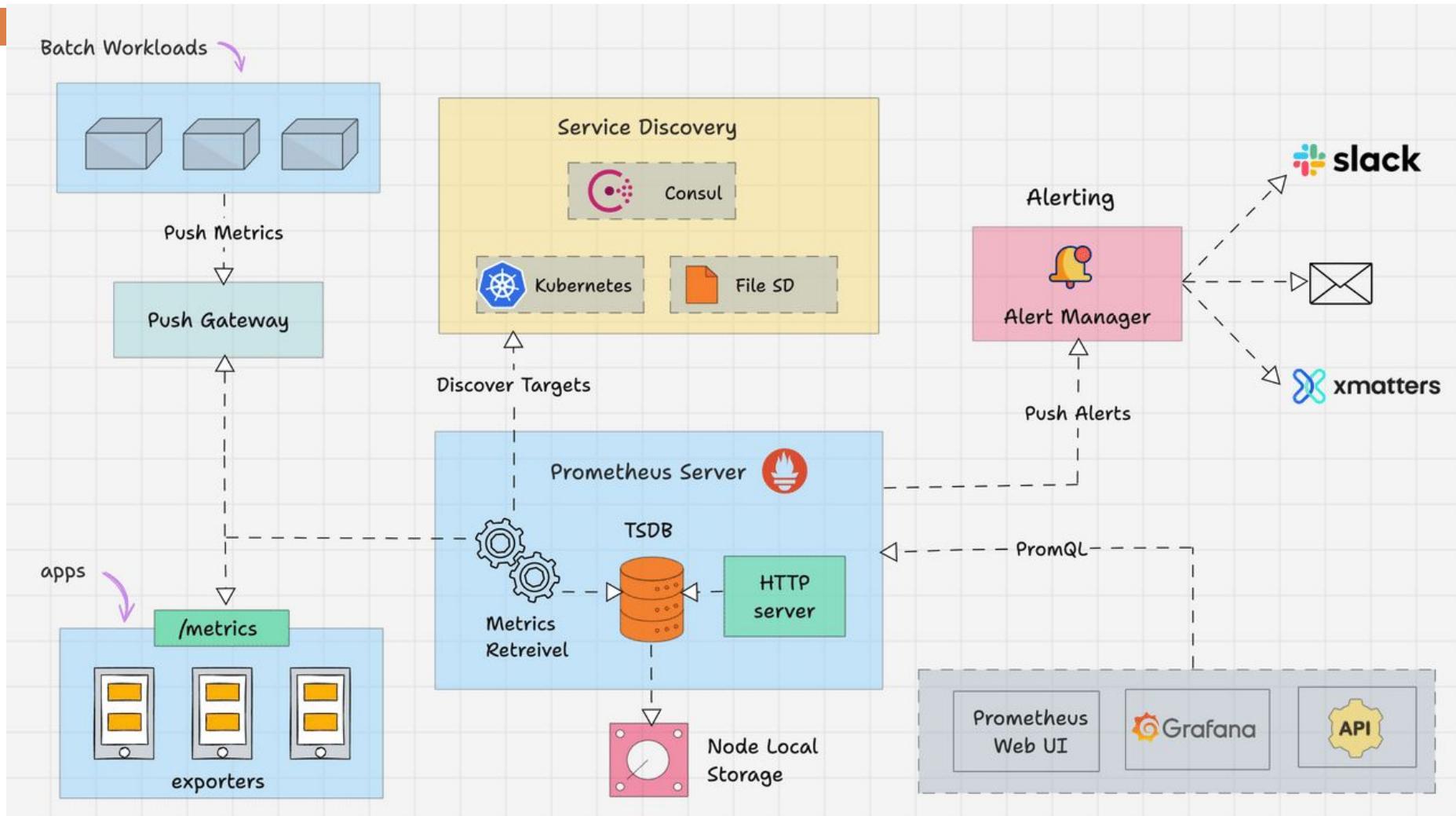
Tool	What	Why	Where	Model	Focus
Prometheus	Monitoring tool	Monitors system performance, resource utilization, and application health	Servers, containers, applications	Open-source	Monitoring
ELK Stack	Logging & Analytics Platform	Collects, stores, analyzes, and visualizes log data	Applications, servers, network devices	Open-source	Logging & Analytics
Grafana	Visualization tool	Creates dashboards and visualizations for monitoring data	Primarily with Prometheus and other data sources	Open-source	Visualization
Nagios	Monitoring tool	Monitors servers, applications, networks, and infrastructure	Servers, applications, network devices	Open-source (Nagios Core) & Commercial	Monitoring
Zabbix	Monitoring tool	Monitors availability, performance, and configuration of servers, networks, and applications	Servers, applications, network devices, cloud environments	Open-source	Monitoring
Datadog	Monitoring & Security Platform	Monitors infrastructure, applications, and security	Servers, containers, applications, cloud environments	Commercial	Monitoring & Security
New Relic	Monitoring tool	Monitors application performance, user experience, and infrastructure	Applications, servers, cloud environments	Commercial	Monitoring
Dynatrace	Application Performance Management (APM) tool	Monitors application performance, user experience, and infrastructure	Applications, servers, cloud environments	Commercial	Monitoring (APM focus)
Splunk	Security & Observability Platform	Collects, analyzes, and visualizes machine-generated data for monitoring, security, and operational intelligence	Applications, servers, network devices, security systems	Commercial	Logging & Analytics (Security focus)
LogicMonitor	Monitoring tool	Monitors infrastructure, applications, and cloud environments	Servers, applications, cloud environments	Commercial	Monitoring

# Logging & Monitoring



# What is Prometheus Architecture?

source:<https://devopscube.com/>



# Grafana





Grafana



Prometheus

# Prometheus & Grafana

