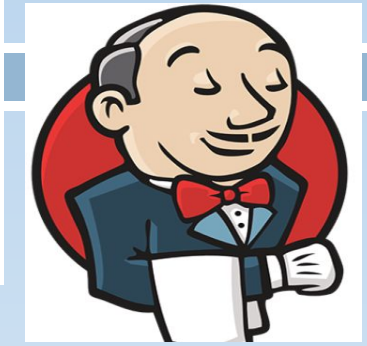
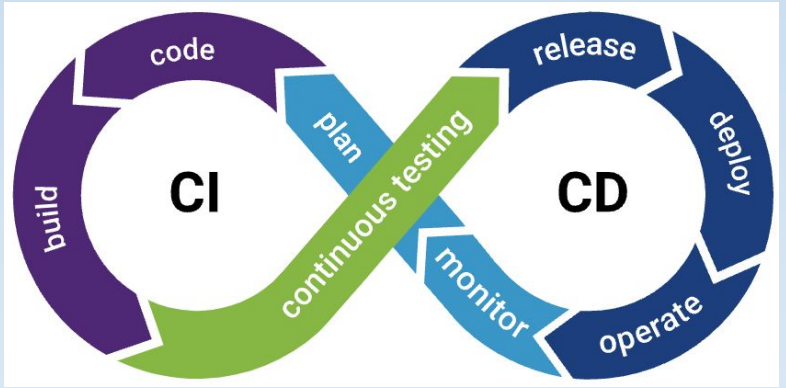


DevOps Atölyesi



Hakan BAYRAKTAR

Jenkins ile 3-Tier Application Deploy (Django App – Docker on EC2)

- Jenkins Nedir? Temel Kavramlar
- 3-Tier Application Mimarisi
- EC2, Autoscaling, Load Balancer ,RDS Servisleri Hakkında Temel Bilgiler
- Terraform

- **Uygulamalı Proje:** Jenkins ile Django + React Uygulamasının AWS EC2 Sunucularına 3-Tier CI/CD Pipeline ile Dağıtılması

Jenkins

Jenkins Nedir

Jenkins: Otomasyonun Kalbi

- Java ile yazılmış açık kaynak CI/CD aracı
- Kodunuzu otomatik test, build ve deploy eder
- Yazılım teslimat sürecini hızlandırır
- Plugin desteği ile her şeye entegre olabilir

Jenkins Neden Bu Kadar Yaygın?

Jenkins'in Popülaritesinin Sebepleri:

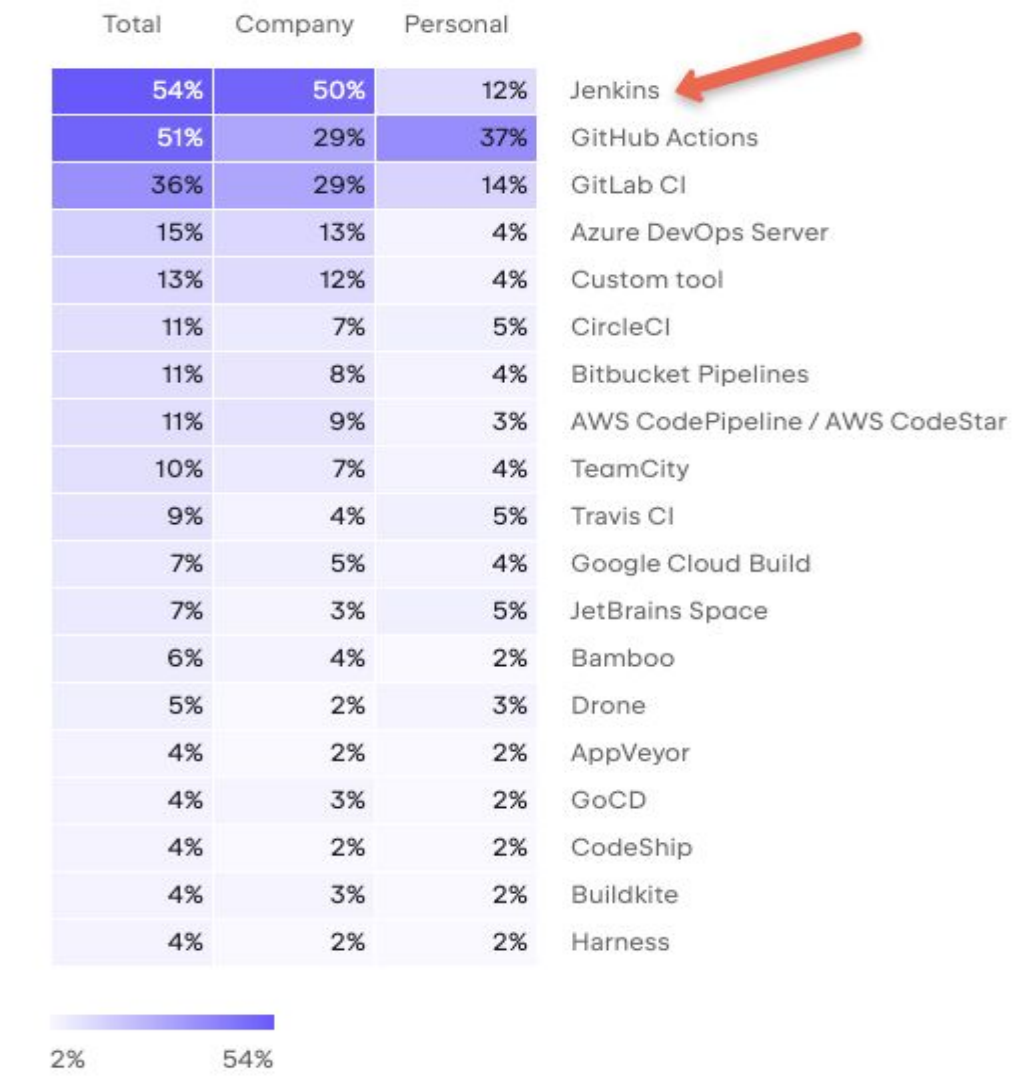
✅ Çok sayıda plugin ile her araca entegre olur

✅ Her dili, her platformu destekler

✅ Geniş topluluk → bol kaynak, hızlı çözüm

✅ Kodu versiyonlanabilir hale getirir

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



Total	Company	Personal	
54%	50%	12%	Jenkins
51%	29%	37%	GitHub Actions
36%	29%	14%	GitLab CI
15%	13%	4%	Azure DevOps Server
13%	12%	4%	Custom tool
11%	7%	5%	CircleCI
11%	8%	4%	Bitbucket Pipelines
11%	9%	3%	AWS CodePipeline / AWS CodeStar
10%	7%	4%	TeamCity
9%	4%	5%	Travis CI
7%	5%	4%	Google Cloud Build
7%	3%	5%	JetBrains Space
6%	4%	2%	Bamboo
5%	2%	3%	Drone
4%	2%	2%	AppVeyor
4%	3%	2%	GoCD
4%	2%	2%	CodeShip
4%	3%	2%	Buildkite
4%	2%	2%	Harness

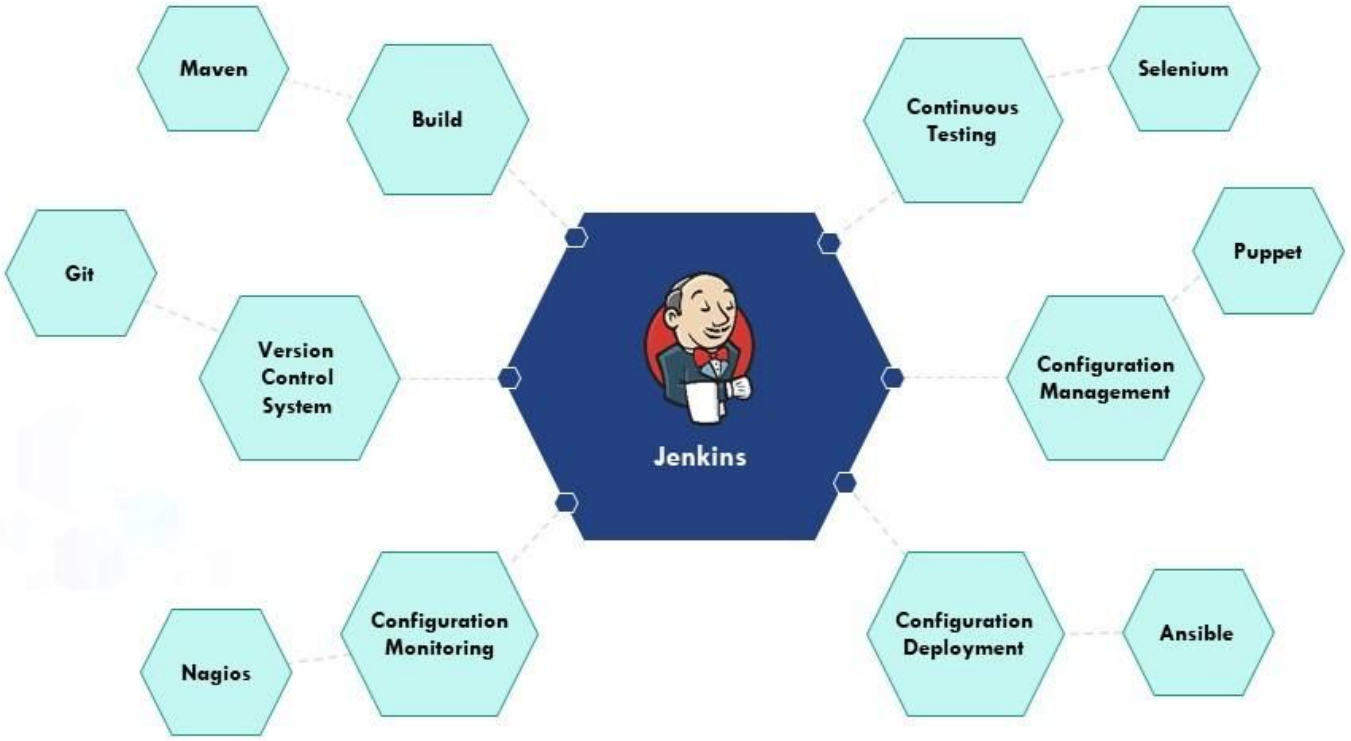
Source: https://devopscube.com/

Jenkins'in Kullanım Alanları

- Uygulama ve altyapı kodları için **Sürekli Entegrasyon (CI)**
- Farklı ortamlara **Sürekli Dağıtım (CD)**
- **IaC** (Terraform, Ansible) entegrasyonu
- Otomatik test, bildirim, yedekleme, cleanup gibi işler

💡 Kısacası: yazılımla ilgili tüm otomasyon Jenkins üzerinden yönetilebilir.

Continuous Integration in Jenkin

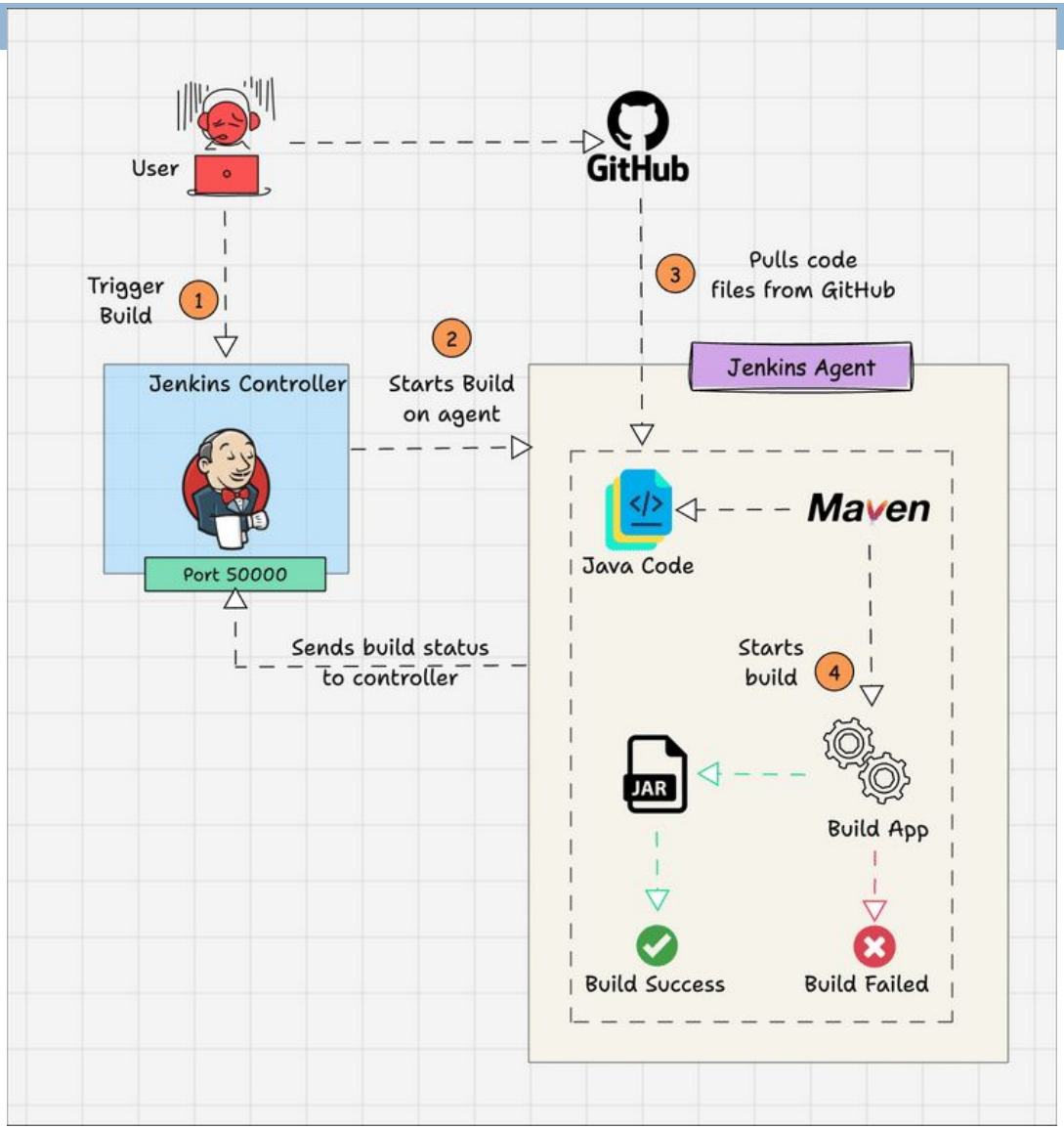


Jenkins CI

🖥️ **Controller (Master)**
Yönetim paneli, job tetikleyici, queue kontrolü

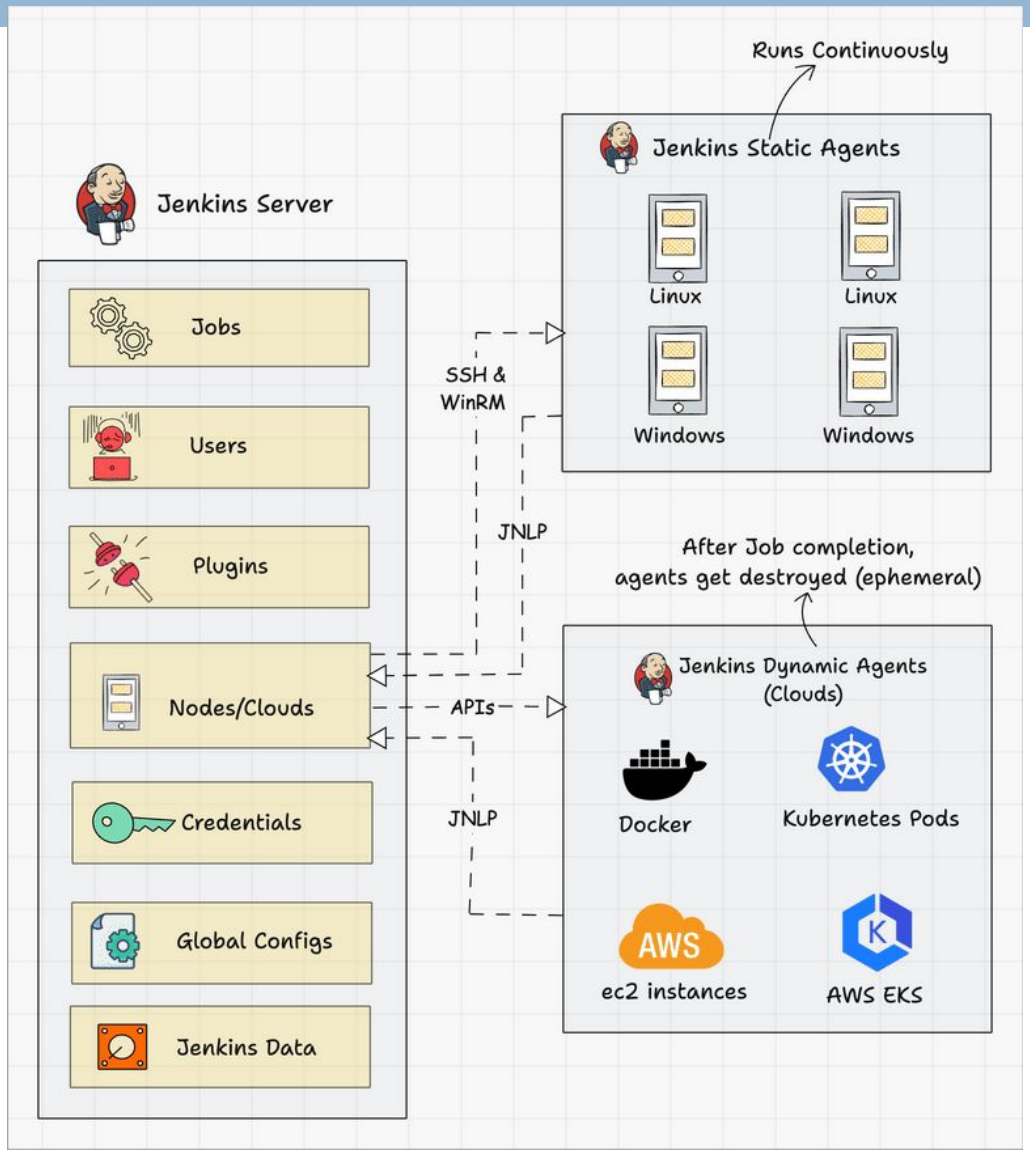
⚙️ **Agent (Node)**
İşleri gerçekten yürüten worker makineler

🔁 Controller + Agent yapısı sayesinde dağıtık ve ölçeklenebilir CI/CD altyapısı sağlanır



Source: https://devopscube.com/

Jenkins Architecture



Source: https://devopscube.com/

Jenkins Temel Kavramlar-1

Job: Jenkins'te her iş bir job'dır (build, deploy vs.)

Build: Job'un her çalıştırılması

Pipeline: Süreci tanımlayan kod dizisi

- Declarative: sade, okunabilir
- Scripted: Groovy tabanlı, esnek

Node: İşlerin yürütüldüğü makine (agent olabilir)

Agent: Spesifik bir iş için kullanılan çalışma ortamı

Plugin: Jenkins'i özelleştiren modüller

Jenkins Temel Kavramlar-2

Workspace: Her job için Jenkins'in oluşturduğu çalışma klasörü

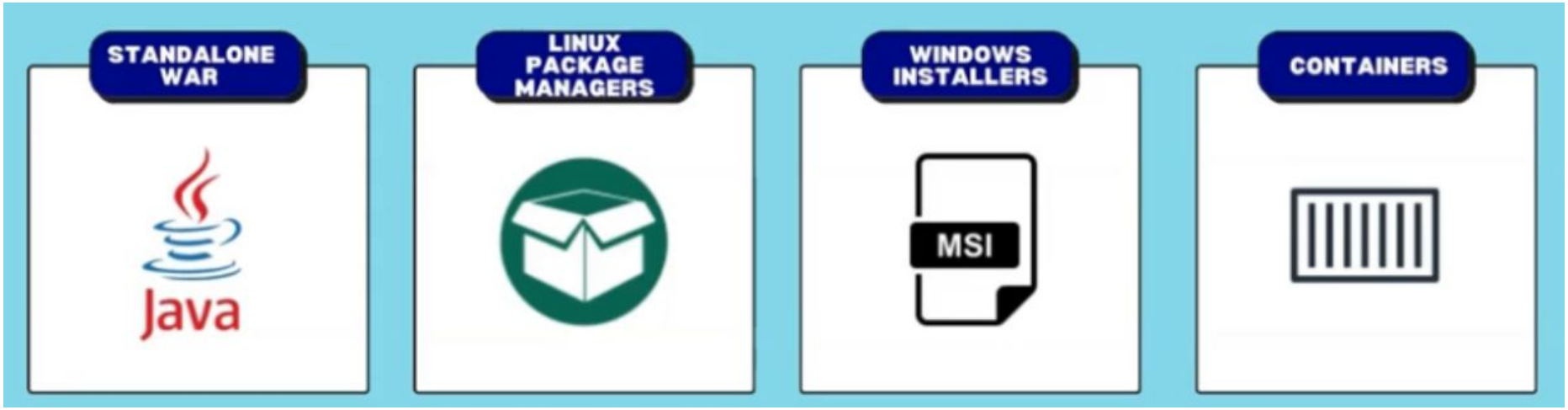
Artifact: Build sonrası oluşan çıktılar

View: Projelerin toplandığı görünümler

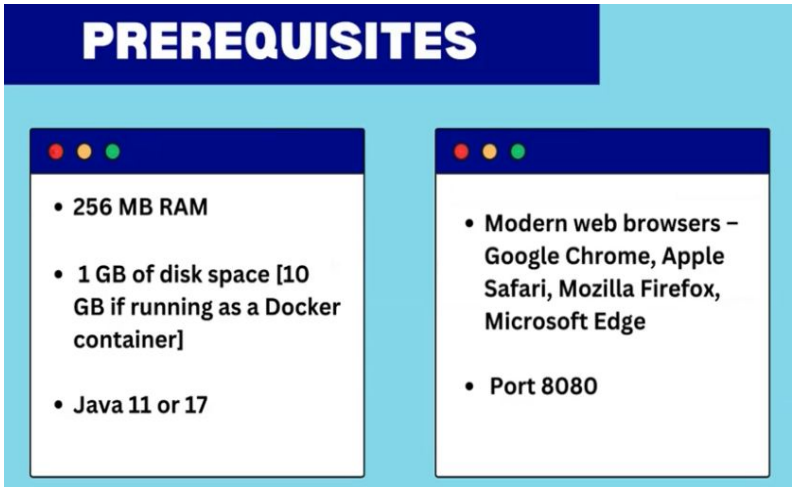
Credential: Güvenli kimlik bilgileri (AWS key, token, şifre vs.)

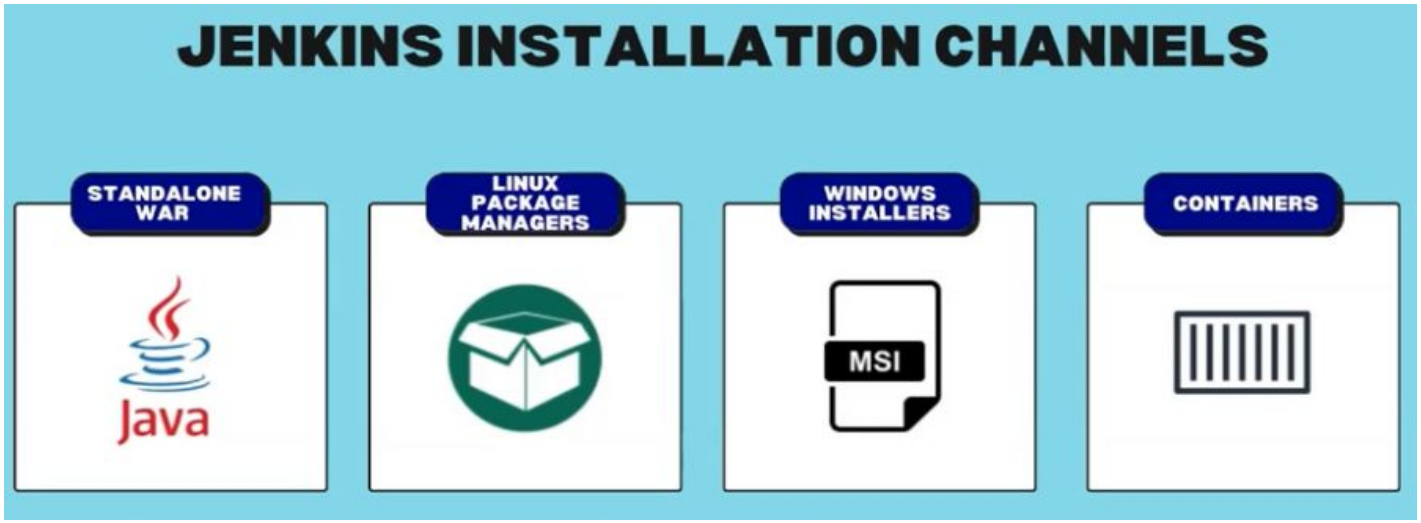
Trigger: SCM değişikliği, zamanlama, manuel tetikleyici

Jenkins installation channels



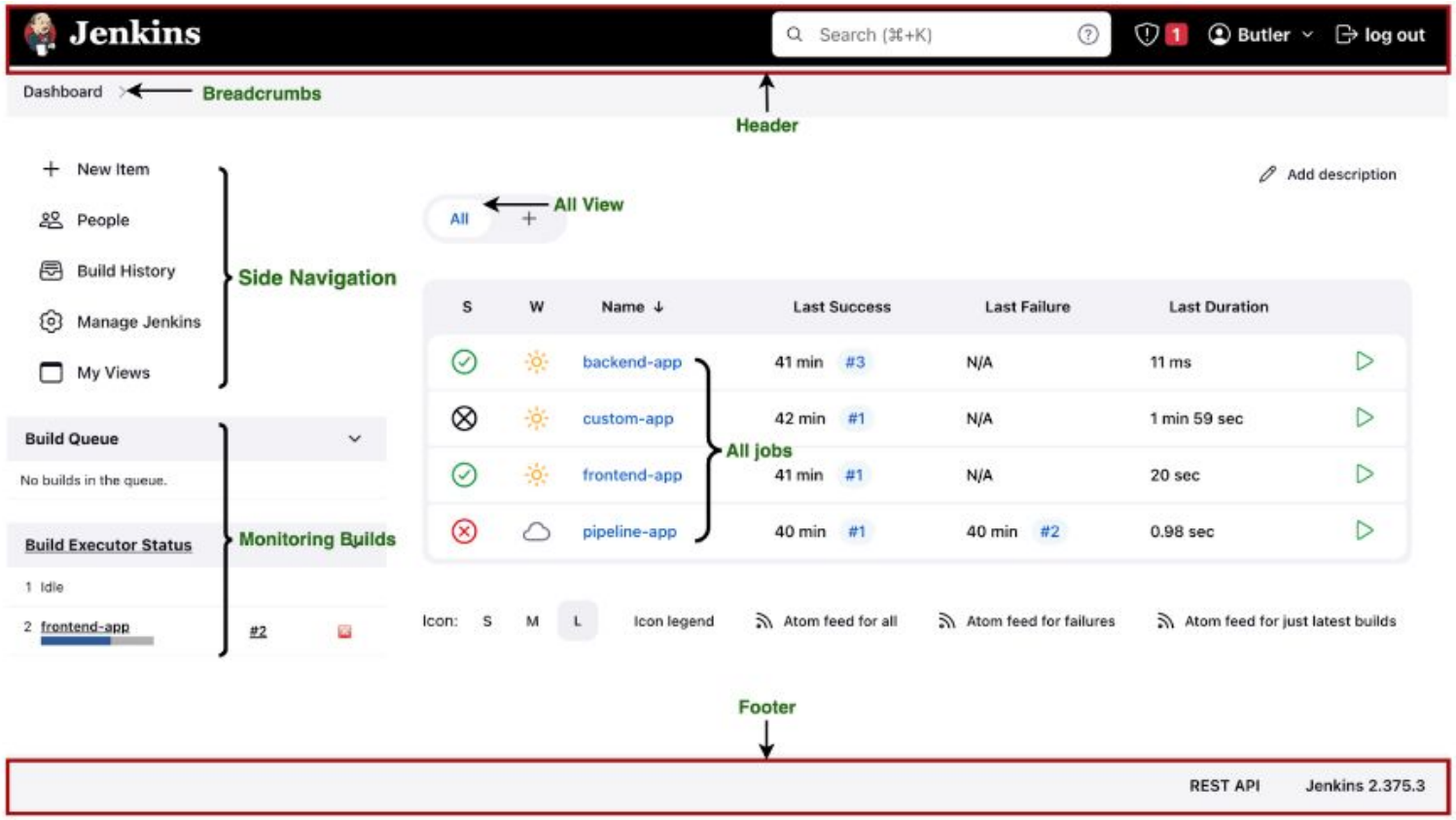
PREREQUISITES





JENKINS Installation

JENKINS Dashboard



Jenkins Plugin'leri Neden Önemli?

🔌 **Plugin = Güç**

- Jenkins’in en büyük gücü eklenti (plugin) desteğidir
- Kod yönetimi, test araçları, bulut servisleriyle entegrasyon sağlar
- Kullanıcı ihtiyaçlarına göre Jenkins’i esnetir ve özelleştirir

En Temel Jenkins Plugin'leri

- **Pipeline** – CI/CD akışlarını kodlamak için temel eklenti
- **Git Plugin** – Git repo bağlantısı için
- **Credentials Plugin** – Gizli bilgi (token, şifre) yönetimi
- **Blue Ocean** – Modern arayüz (gelişmiş UI)
- **Job DSL** – Job'ları kodla tanımlamak için
- **Parameterized Trigger** – Diğer job'ları tetiklemek için
- **Slack Notification** – Bildirimler için
- **Docker Pipeline** – Docker komutlarını çalıştırmak için

Jenkins Pipeline Nedir?

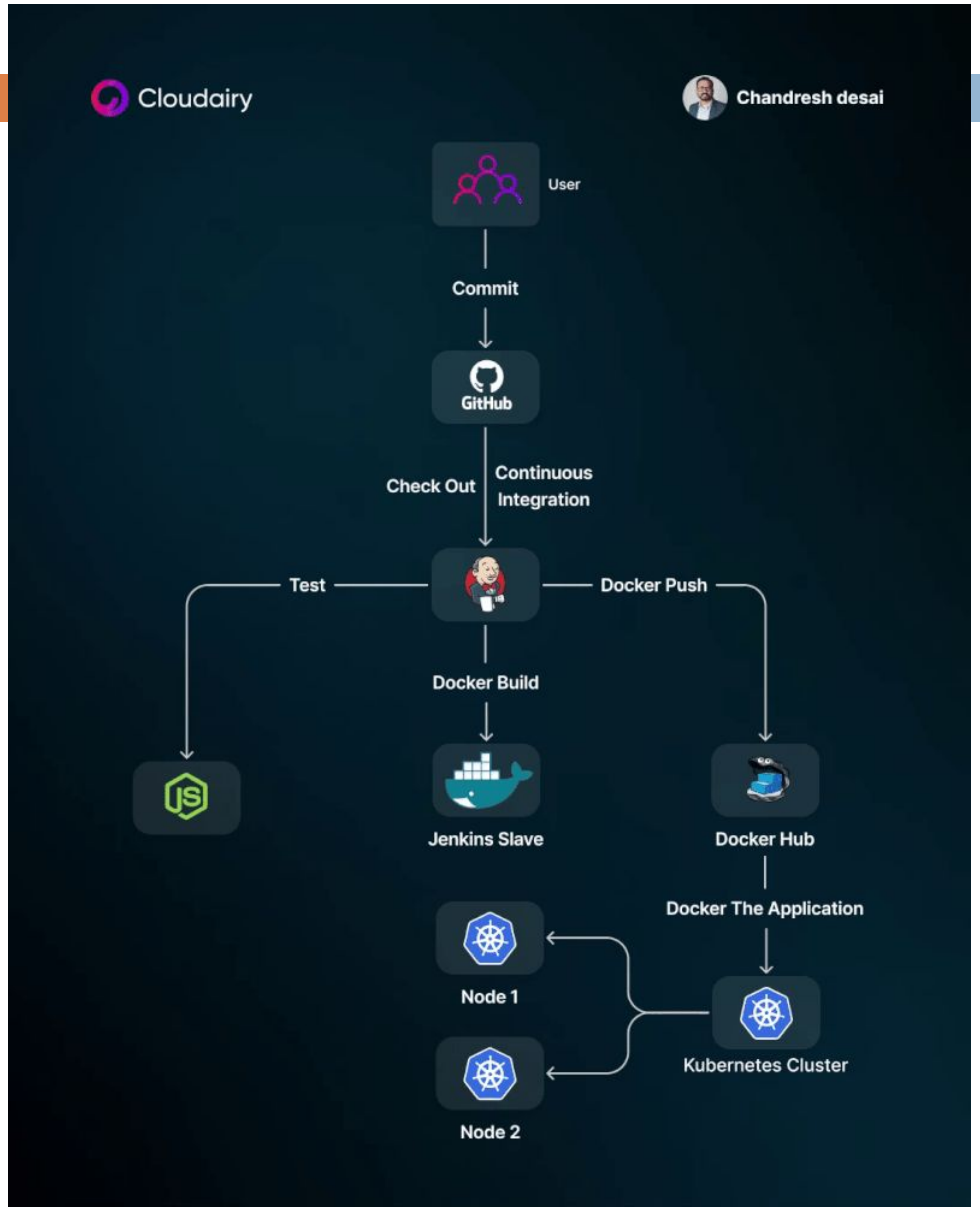
🧩 **Pipeline** = CI/CD sürecini kod ile tarif etmek

- Kodun test edilmesi
- Docker image oluşturulması
- Otomatik deploy
- Versiyonlama, koşul, paralel işlem gibi adımların tanımı

Pipeline türleri:

- Declarative: Kolay ve okunabilir
- Scripted: Groovy ile tam kontrol

Pipeline Yapısı




```
pipeline {
  agent any
  environment {
    VAR = "value"
  }
  stages {
    stage('Build') {
      steps {
        echo "Building..."
      }
    }
  }
  post {
    success {
      echo "İşlem başarılı!"
    }
  }
}
```

Önemli Pipeline Komutları

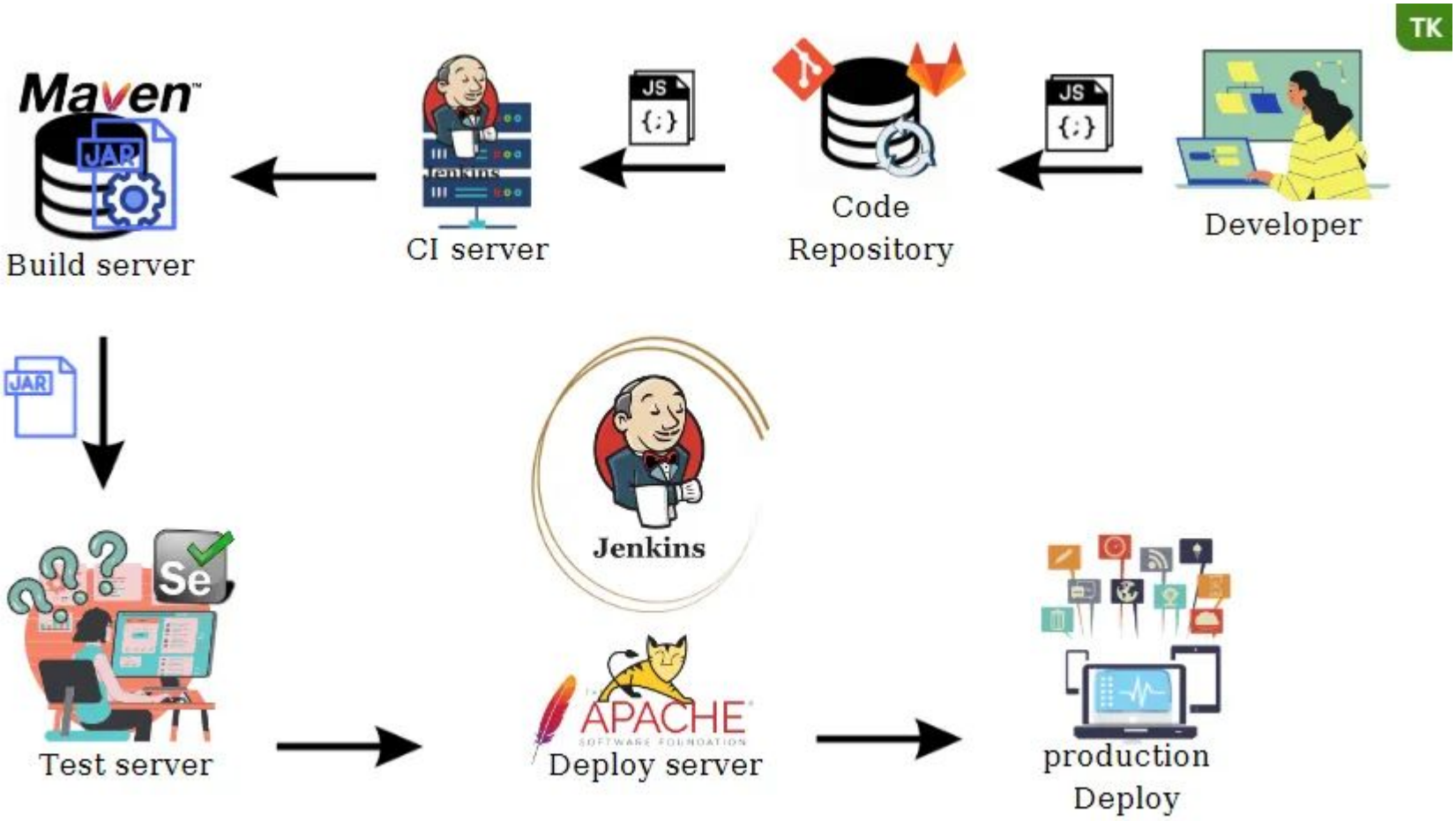
Blok	Açıklama
`pipeline {}`	Tüm yapıyı tanımlar
`agent`	Nerede çalışacak?
`stages`	Ana aşamalar
`steps`	Her aşamada yapılacak işlemler
`environment`	Ortam değişkenleri
`post`	Başarı/başarısızlık sonrası ne olacak?

Gelişmiş Pipeline Yapıları

- `when` → Koşullu çalıştırma (örn. sadece “main” branch)
- `parameters` → Kullanıcıdan veri alma
- `input` → Onay almadan devam etme
- `parallel` → Aynı anda birden fazla job çalıştırma

🎯 Gerçek CI/CD senaryolarında bu yapıların çoğu birlikte kullanılır

Jenkins pipeline



Pipeline kurarken aşağıdaki sorulara cevap vermemiz gerekir

1- Kod nerede depolanacak

2- Nasıl build edeceğim

4- 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
            }
        }
    }
}
```

Parametre örneği Pipeline

groovy Copy code

```groovy
pipeline {
    agent any
    parameters {
        string(name: 'ENVIRONMENT', defaultValue: 'staging', description: 'Deploy e
    }

    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

```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'
      }
    }
  }
}
```

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

EC2

Amazon EC2

- EC2 is one of the most popular of AWS' offering
- EC2 = Elastic Compute Cloud
- It mainly consists in the capability of :
  - Renting virtual machines (EC2)
  - Storing data on virtual drives (EBS)
  - Distributing load across machines (ELB)
  - Scaling the services using an auto-scaling group (ASG)
- Knowing EC2 is fundamental to understand how the Cloud works

EC2 Konfigürasyon Bileşenleri

İşletim Sistemi: Linux / Windows / Mac

CPU, RAM: Hesaplama gücü ve bellek

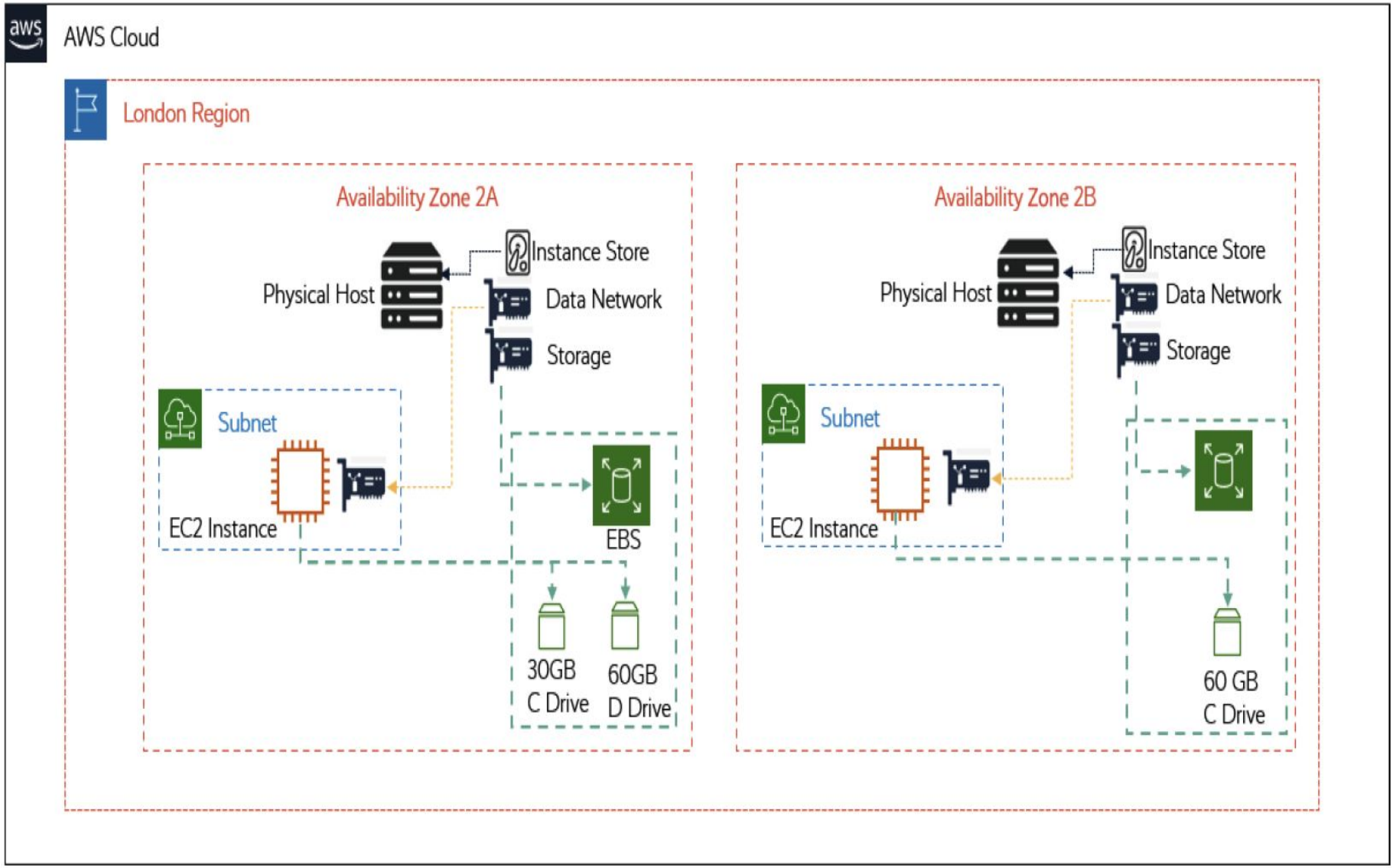
Depolama: EBS (network bağlı) / Instance Store (fiziksel disk)

Ağ Kartı: IP adresi, ağ hızı

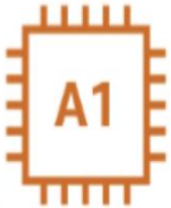
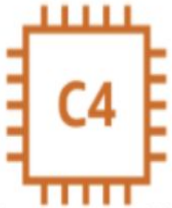
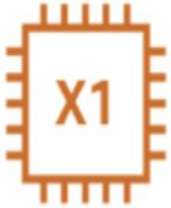
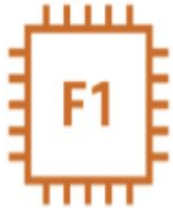
Güvenlik Grubu: EC2 için firewall kuralları

User Data: İlk çalıştırma betiği (örneğin: docker kur)

EC2 instance components



AWS EC2 INSTANCE TYPES

General Purpose	Compute Optimised	Memory Optimised	Accelerated Computing	Storage Optimised
 ARM based core and custom silicon	 Compute - CPU intensive apps and DBs	 RAM - Memory intensive apps and DB's	 Processing optimised- Machine Learning	 High Disk Throughput - Big data clusters
 Tiny - Web servers and small DBs		 Xtreme RAM - For SAP/Spark	 Graphics Intensive - Video and streaming	 IOPS - NoSQL DBs
 Main - App servers and general purpose		 High Compute and High Memory - Gaming	 Field Programmable - Hardware acceleration	 Dense Storage - Data Warehousing

https://instances.vantage.sh/

Buying EC2 Options

On-Demand — Least Commitment

- low cost and flexible
- only pay per hour
- short-term, spiky, unpredictable workloads
- cannot be interrupted
- For first time apps

Spot upto 90% — Biggest Savings

- request spare computing capacity
- flexible start and end times
- Can handle interruptions (server randomly stopping and starting)
- For non-critical background jobs

Reserved upto 75% off — Best Long-term

- steady state or predictable usage
- commit to EC2 over a 1 or 3 year term
- Can resell unused reserved instances

Dedicated — Most Expensive

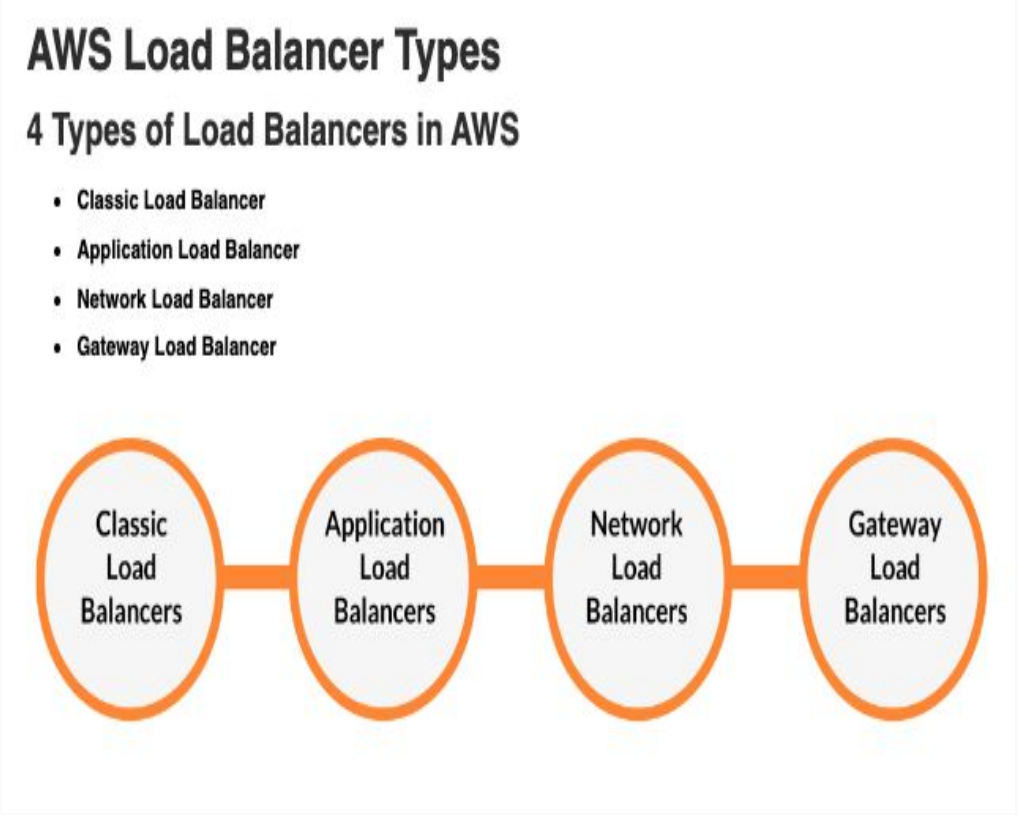
- Dedicated servers
- Can be on-demand or reserved (upto 70% off)
- When you need a guarantee of isolate hardware (enterprise requirements)

High Availability And Scalability

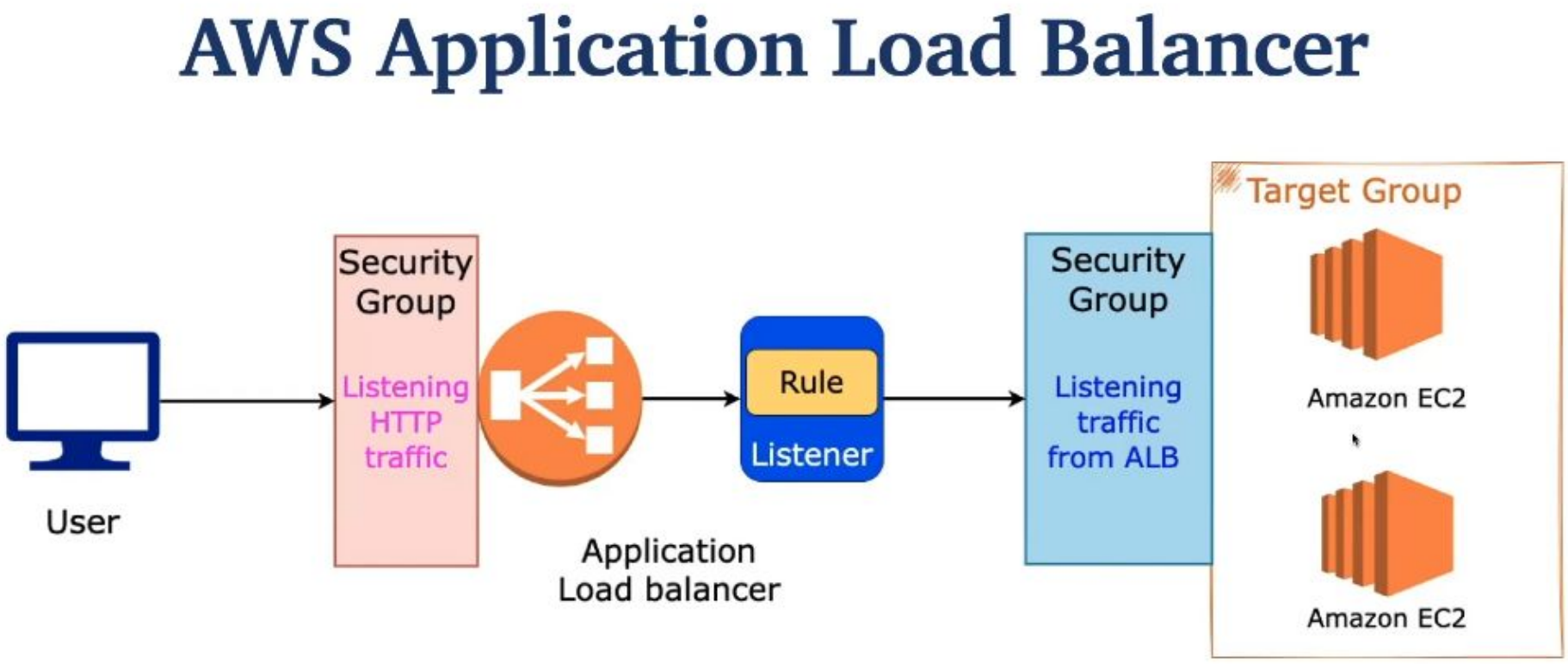
AWS'de Yük Dağılımı (Load Balancing)

Elastic Load Balancer (ELB)

- Trafiği birden fazla EC2 instance'a dağıtır
- Uygulamanın erişilebilirliğini ve güvenilirliğini artırır



ALB



Application Load Balancer (ALB)

- URL bazlı routing (örnek: `/api`, `/admin`)
- Health check ile sağlıklı instance'lara yönlendirme yapar

ELB

Feature	Application Load Balancer	Network Load Balancer	Classic Load Balancer
Protocols	HTTP, HTTPS	TCP	TCP, SSL/TLS, HTTP, HTTPS
Platforms	VPC	VPC	EC2-Classic, VPC
Health checks	✓	✓	✓
CloudWatch metrics	✓	✓	✓
Logging	✓	✓	✓
Path-Based Routing	✓		
Host-Based Routing	✓		
Native HTTP/2	✓		
			✓
SSL offloading	✓		✓
Static IP		✓	
Elastic IP address		✓	
Slow start	✓		

Auto Scaling Group (ASG)

What is Amazon EC2 Auto Scaling?

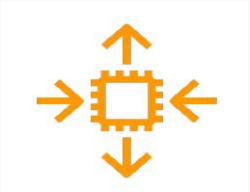
Amazon EC2 Auto Scaling is a region-specific service used to maintain application availability and enables users to automatically add or remove EC2 instances according to the compute workloads.

EC2 Auto Scaling supports automatic Horizontal Scaling (increases or decreases the number of EC2 instances) rather than Vertical Scaling (increases or decreases EC2 instances like large, small, medium).

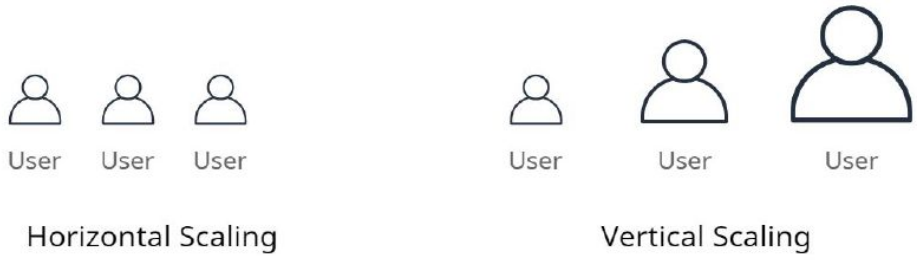
The Auto Scaling group is a collection of the minimum number of EC2 used for high availability.

we can configure automatic scaling for all of the scalable resources :

- **Amazon EC2**
- **Amazon EC2 Spot Fleets**
- **Amazon ECS**
- **Amazon DynamoDB**
- **Amazon Aurora**



It scales across multiple Availability Zones within the same AWS region.



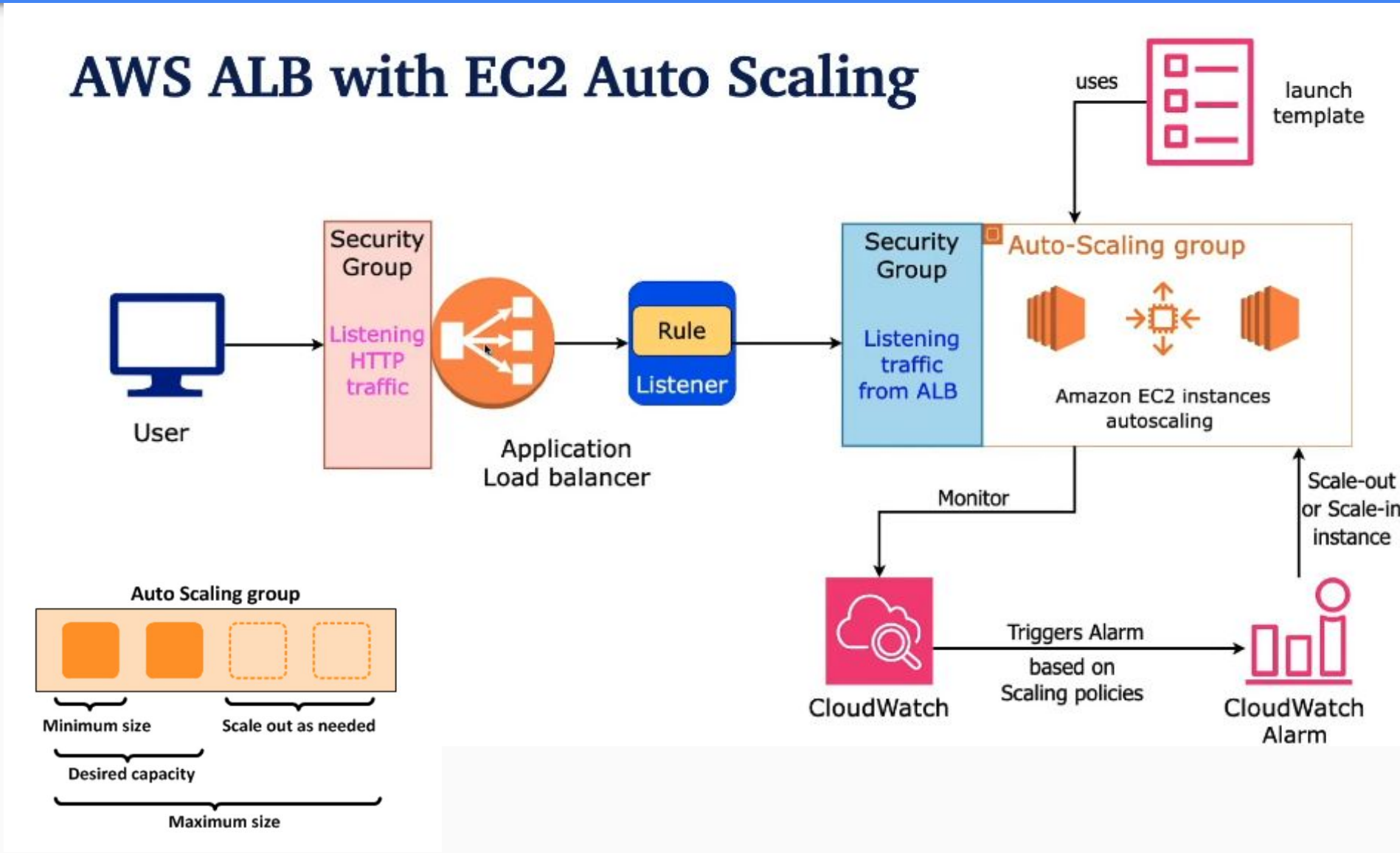
ASG Yapı Taşları

Launch Template: EC2'ların hangi konfigürasyonla başlayacağını belirler
Scaling Policy: Ne zaman yeni instance başlatılacağını tanımlar

Target Group: Hangi Load Balancer'a bağlanacağını belirler

Health Check: Bozuk EC2 otomatik kaldırılır

EC2 Auto Scaling



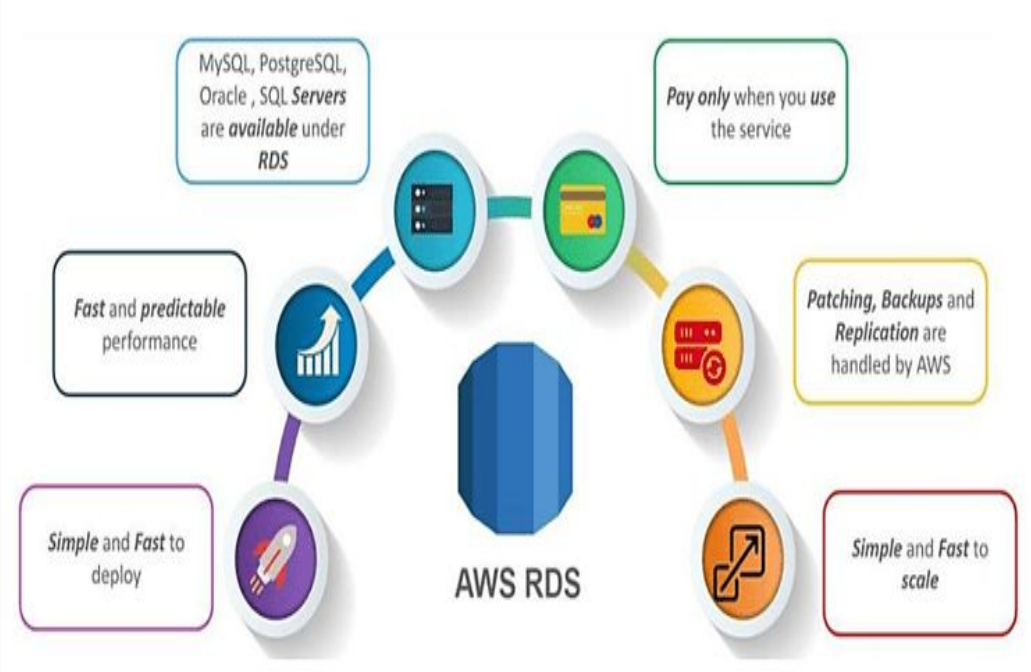
RDS

Amazon RDS Nedir?

RDS (Relational Database Service): Yönetilen bir veritabanı hizmetidir.

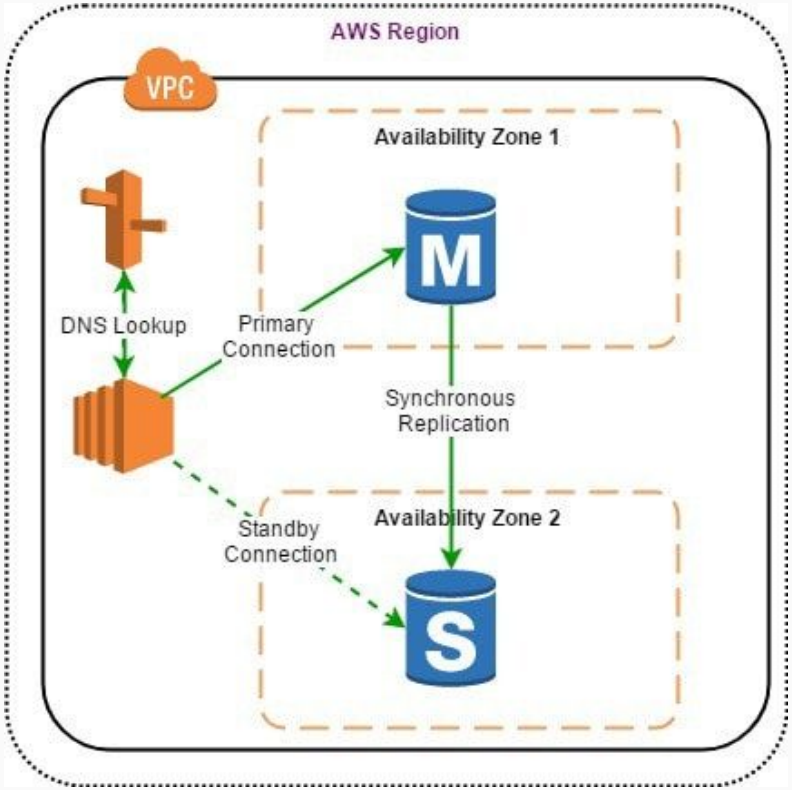
- Yedekleme, güncelleme, ölçekleme gibi işleri AWS otomatik yapar
- MySQL, PostgreSQL, MariaDB, Oracle, SQL Server destekler
  Private subnet içinde barındırılarak dış erişim engellenebilir

What are the benefits of Amazon RDS?

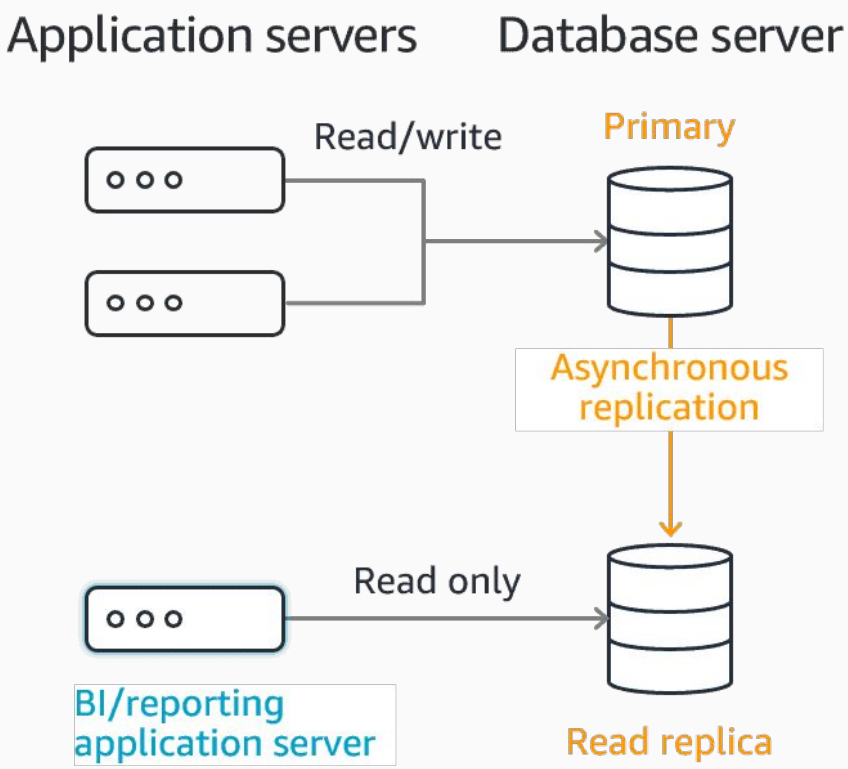


AWS RDS Has Two Main Features:

Multi-AZ Deployment: Yüksek erişilebilirlik için yedek DB



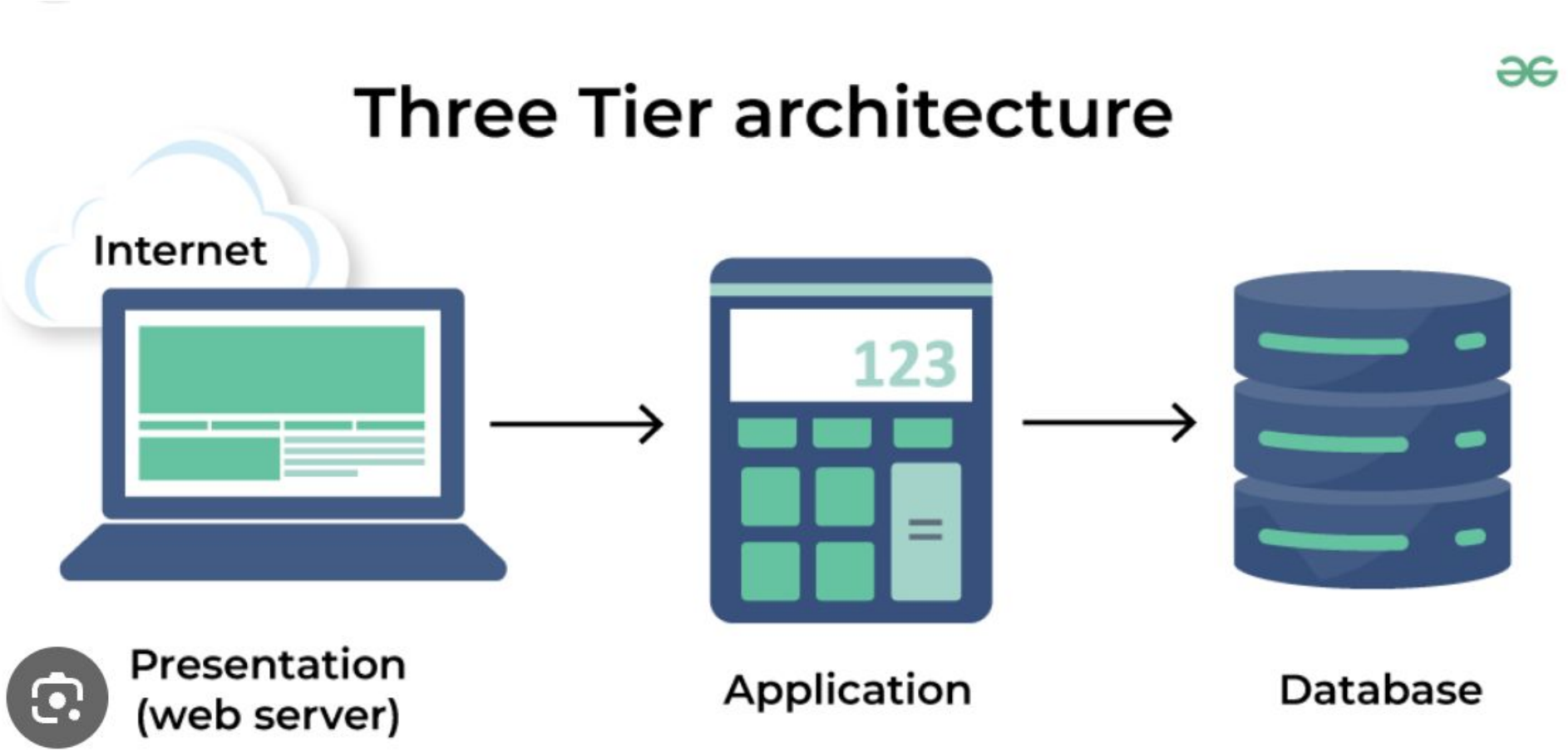
Read Replica: Okuma performansını artırmak için çoğaltma



Snapshot: Manuel veya otomatik yedek alma
Encryption: Veritabanı ve yedekleri şifreleyebilir

3-Tier Mimari

3-Tier Mimari Nedir?



3-Tier Mimari Nedir?

🎯 **3-Tier Mimari**, uygulamayı 3 ayrı katmana bölerek her bileşenin izole ve kontrollü çalışmasını sağlayan modern bir sistem mimarisidir:

- **1. Katman – Frontend (Sunum Katmanı)**

- Kullanıcının doğrudan etkileşim kurduğu arayüzdür
- React, Angular, HTML/CSS ile geliştirilir

- **2. Katman – Backend (Uygulama Katmanı)**

- İş mantığını ve API'leri içerir
- Django, Node.js gibi framework'lerle kurulur

- **3. Katman – Veritabanı (Data Katmanı)**

- Kalıcı verilerin saklandığı yerdir
- MySQL, PostgreSQL, MongoDB gibi sistemler kullanılır

Neden 3 Katman?

- 🔐 **Güvenlik:**
  Her katman izole edilir → sadece ihtiyacı olan katman erişebilir (örneğin: frontend veritabanını doğrudan göremez)
- ⚙️ **Performans:**
  Her katman ayrı optimize edilir → Örneğin frontend statik olarak dağıtılır, backend CPU ağırlıklı çalışır, DB ise ayrı kaynaklarda barınır
- 🔄 **Yönetilebilirlik:**
  Katmanlar bağımsız güncellenebilir → Sadece backend'i değiştirip frontend'e dokunmadan deploy yapılabilir

Neden Tercih Edilir?

💡 3-Tier mimari; büyük, ölçeklenebilir ve sürdürülebilir uygulamalar geliştirmek için en çok tercih edilen yapıdır.

Katman	Ne Sağlar?
Frontend	Kullanıcı deneyimi, arayüz sunar
Backend	Uygulama mantığını yönetir
Veritabanı	Veriyi saklar ve erişim kontrolü sağlar

🔍 Her katman **izole çalıştığı** için sistem daha **güvenli**, **esnek**, **bakımı kolay** ve **performanslı** hale gelir.

TERRAFORM

Terraform Nedir?

Infrastructure as Code (IaC) nedir?

Terraform = HashiCorp tarafından geliştirilen açık kaynaklı IaC aracı

Ne işe yarar?

- **Cloud altyapısını (AWS, Azure, GCP, vs.) kodla tanımlama**
- **Otomatik, güvenli ve tekrar edilebilir kaynak oluşturma**

Neden Terraform?

❗ **Deklaratif yapı** (Ne istendiğini söylersin, nasıl yapılacağını değil)

☁️ **Cloud Provider bağımsızlığı** (AWS, GCP, Azure)

🔁 **Tekrar edilebilirlik ve sürüm kontrolü**

🔐 **Güvenli ve denetlenebilir yapı**

Terraform Temel Yapısı

`.tf` dosyaları (Terraform configuration files)

`provider` → AWS, Azure, vs.

`resource` → Oluşturulacak altyapı nesneleri (EC2, S3, RDS..)

`variable`, `output`, `module` gibi yapı taşları

Terraform Workflow – 4 Temel Adım

1. `terraform init` – Başlatma (provider'ları indir)
2. `terraform plan` – Değişiklikleri göster
3. `terraform apply` – Değişiklikleri uygula
4. `terraform destroy` – Altyapıyı sil

Basit Bir Örnek - AWS EC2

```
provider "aws" {
  region = "us-east-1"
}

resource "aws_instance" "example" {
  ami           =
"ami-0c55b159cbfafe1f0"
  instance_type = "t2.micro"
}
```

Terraform State Dosyası

`terraform.tfstate` – Terraform'un altyapıyı izlediği dosya

Lokal veya Remote (S3 + DynamoDB gibi)

`terraform refresh`, `terraform state list`

Terraform ile Modülerlik

`modules` nedir?

Örnek: VPC, EC2 gibi kaynaklar modül haline getirilerek tekrar kullanılabilir

Büyük altyapılarda düzen sağlar

Terraform Backend - State Yönetimi

Neden önemli?

- Takım çalışmasında state çakışmalarını önler

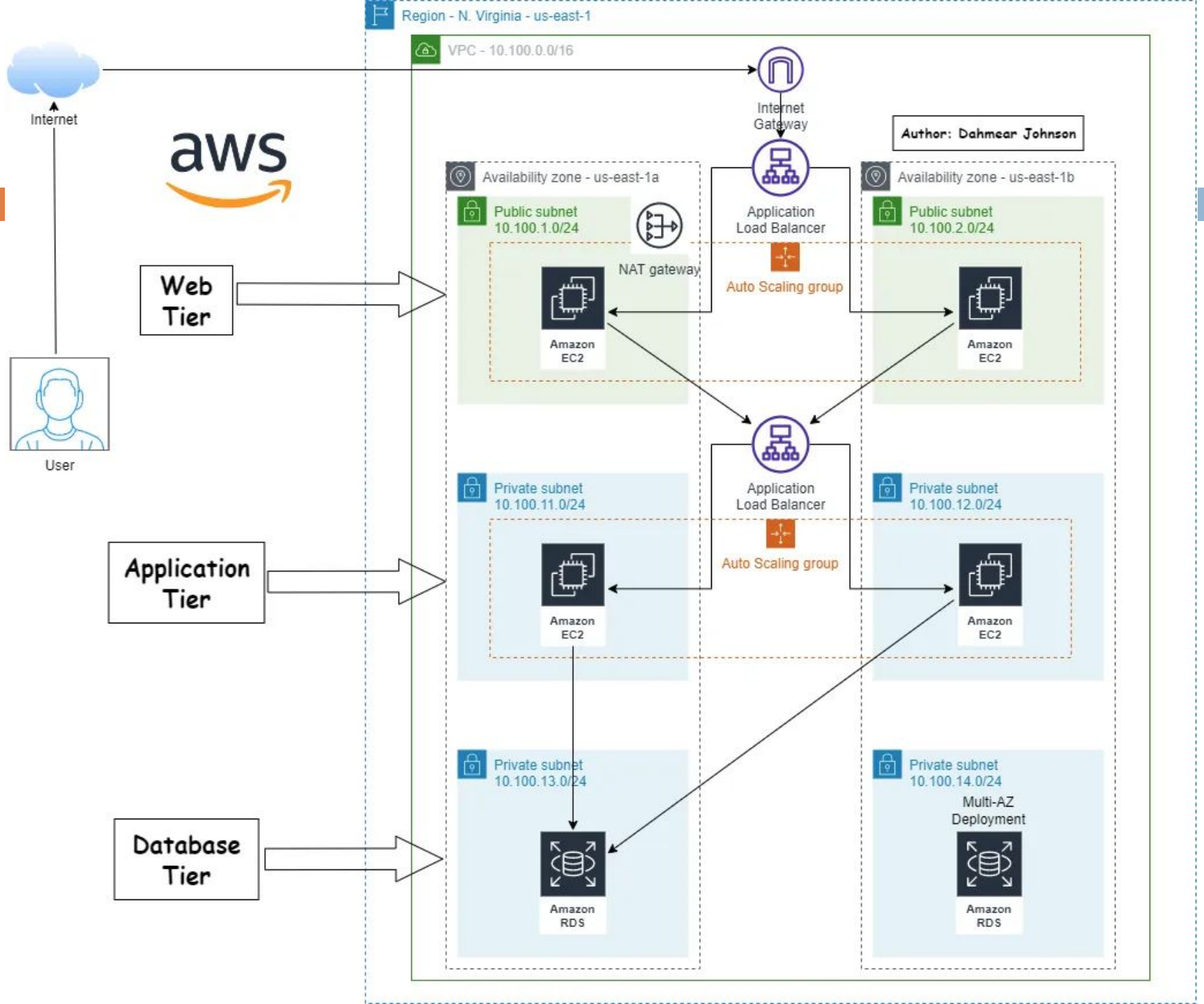
Remote backend: AWS S3, Azure Blob, GCS

Lock mekanizması: DynamoDB (AWS)

Terraform Kullanım Alanları

- AWS altyapı kurulumu
- CI/CD entegrasyonu
- Kubernetes cluster kurulumu
- Multi-cloud altyapı yönetimi

Jenkins ile 3-Tier Application Deploy (Django App – Docker on EC2)


Region - N. Virginia - us-east-1
VPC - 10.100.0.0/16
Internet
Internet Gateway
Author: Dahmear Johnson
aws
Availability zone - us-east-1a
Availability zone - us-east-1b
Public subnet 10.100.1.0/24
Public subnet 10.100.2.0/24
NAT gateway
Application Load Balancer
Auto Scaling group
Amazon EC2
Amazon EC2
Web Tier
User
Application Load Balancer
Auto Scaling group
Private subnet 10.100.11.0/24
Private subnet 10.100.12.0/24
Amazon EC2
Amazon EC2
Application Tier
Private subnet 10.100.13.0/24
Private subnet 10.100.14.0/24
Multi-AZ Deployment
Amazon RDS
Amazon RDS
Database Tier

Proje Adımları – 3-Tier CI/CD Pipeline (Terraform + Jenkins)

Remote Backend Yapılandırması
→ Terraform için S3 ve DynamoDB oluşturuldu (state yönetimi)
Temel Altyapı Kurulumu
→ Terraform ile VPC, subnet'ler, bastion host, security group'lar oluşturuldu
Jenkins EC2 Üzerine Kuruldu
→ EC2 instance'a Jenkins yüklendi, plugin'ler ve credentials ayarlandı
Jenkins Pipeline'ları Tanımlandı
→ `infra-create`, `docker-deploy`, `destroy-aws-infra` job'ları oluşturuldu
Uygulama Katmanı Kurulumu
→ Backend (Django) ve Frontend (React), Docker ile hazırlanıp EC2'lara dağıtıldı
→ Auto Scaling Group + Launch Template kullanıldı
ALB ve Target Group Yapılandırması
→ Trafik dengesi için ALB kuruldu, hedefler EC2 instance'lara bağlandı
RDS (MySQL) Kurulumu
→ Private subnet içinde yalıtılmış veritabanı oluşturuldu
CI/CD Otomasyonu Tamamlandı
→ Her kod değişikliğinde `docker-deploy` job'ı ile otomatik build & deploy sağlandı