T.C. SANAYİ VE TEKNOLOJİ BAKANLIĞI

#MİLLİ TEKNOLOJİ HAMLESİ

TÜBİTAK

MODERN MİMARİLERDE DEVOPS/DEVSECOPS SÜREÇLERİ

ARAŞTIRMA SERİSİ - SAYI 16

BİLGEM

YAZILIM TEKNOLOJİLERİ ARAŞTIRMA ENSTİTÜSÜ

Simge ve Kısaltmalar

Kısaltmalar	Açıklama
TÜBİTAK	Türkiye Bilimsel ve Teknolojik Araştırma Kurumu
BİLGEM	Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi
YTE	Yazılım Teknolojileri Araştırma Enstitüsü
API	Uygulama Programlama Arayüzü (Application Programming Interface)
CI/CD	Sürekli Entegrasyon (Continuous Integration – CI) / Sürekli Dağıtım (Continuous Deployment – CD)
CIS	İnternet Güvenliği Merkezi (Center for Internet Security)
CNCF	Cloud Native Computing Foundation
DAST	Dinamik Uygulama Güvenlik Testi (Dynamic Application Security Testing)
DevOps	Geliştirme (Development) - Operasyon (Operations)
DevSecOps	Geliştirme (Development) - Güvenlik (Security) - Operasyon (Operations)
IaC	Kod Olarak Altyapı (Infrastructure as Code)
IDE	Entegre Geliştirme Ortamı (Integrated Development Environment)
NIST	Ulusal Standartlar ve Teknoloji Enstitüsü (National Institute of Standards and Technology)
OWASP	Açık Web Uygulama Güvenliği Projesi (Open Web Application Security Project)
RBAC	Rol Tabanlı Erişim Kontrolü (Role-Based Access Control)
SAST	Statik Uygulama Güvenlik Testi (Static Application Security Testing)
SCA	Yazılım Kompozisyon Analizi (Software Composition Analysis)

Yazar
Ayşegül ÖZKAYA EREN

Yayın Koordinatörü
Kübra ERTÜRK

Editörler
İbrahim Haluk AVCI
Beyza ŞENEL
Tuğçe YILMAZ

Tasarım
Şeyma KOÇER

İletişim: 0(312) 289 92 22 - yte.bilgi@tubitak.gov.tr

https://bilgem.tubitak.gov.tr/yte/

İçindekiler

Ön Söz

TÜBİTAK BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE), 2012 yılından bu yana yazılım teknolojilerinde Ar-Ge faaliyetleri yürüten bir araştırma kuruluşudur. Araştırma faaliyetlerinde elde ettiği birikimini stratejik, hassas ve kritik projeler yürüterek kamu adına hayata geçirmekte; kurumlarımıza dijital dönüşüm, yazılım geliştirme teknolojileri ve kalite süreçleri konusunda danışmanlık vermektedir.

TÜBİTAK BİLGEM YTE tarafından hazırlanan Araştırma Serisi ile kurum içi içerik üretme çalışmalarının yaygınlaştırılması ve hazırlanan içeriklerin sektörün erişimine açılması amaçlanmaktadır. Araştırma Serisi'nde yayınlanan çalışmalar TÜBİTAK BİLGEM YTE çalışanlarının projelerde elde ettiği bilgi birikimini paylaşmak adına derlenmiştir. Bu çalışmalar ile ülkemizin yazılım sektörüne katkı sağlanılması hedeflenmektedir.

Giriş

Günümüzde gelişen teknoloji ve kullanıcı ihtiyaçlarının sürekli değişmesi, yazılım sektöründe de büyük bir değişimi beraberinde getirmektedir. Hızlı değişen müşteri ihtiyaçları ve rekabetçi ortamın sonucu olarak yazılım organizasyonları, yazılımlarını daha sık güncellemek durumunda kalmaktadır. Ancak, yazılım dünyasında gittikçe fazla özellik içeren karmaşık yazılım mimarilerinin hızlı teslimatını güvenle yapmak zordur.

Yazılım ürünlerinin hızlı ve hatasız dağıtımını yapma gereksinimi sonucu Sürekli Entegrasyon (Continuous Integration - CI) ve Sürekli Dağıtım (Continuous Deployment - CD) süreçleri oluşmuştur. CI/CD süreçlerinin temelini ise, yazılım geliştirme ve dağıtımının hızlı ve güvenli yapılmasını sağlayan otomasyon ve iş birliği içinde çalışma prensipleri oluşturur. Geleneksel süreçlerde, farklı takımların farklı rolleri üstlenmesiyle ilerleyen yazılım geliştirme süreçlerinde, otomasyon ve iş birliği geri planda kalmakta, iletişim eksikliği ve ekiplerin ortak bir dil kullanamaması gibi nedenler verimsizliğe sebep olmaktadır.

Bu problemleri çözmek için, farklı görevlere sahip geliştirici (developer) ve operasyon (operations) ekiplerinin iletişimini güçlendirmeyi amaçlayan DevOps (Development/Operations) kültürü ortaya çıkmıştır. DevOps kültürü, temeli çevik yazılıma dayanan, CI/CD süreçleri uygulanarak hızlı, hatasız ve kaliteli yazılım geliştirme ve dağıtımın yapılmasını sağlayan bir yaklaşım biçimidir.

Bu çalışma kapsamında, DevOps ve DevOps süreçlerine güvenlik katmanının eklenmesini hedefleyen DevSecOps (Development/Security/Operations) süreçleri ve modern mimarilerde bu süreçlerde uygulanması gereken adımlar ve kullanılabilecek açık kaynak araçlardan bahsedilmiştir.

DevOps Nedir?

Yazılım geliştirme ve teslimat süreçleri, farklı ekiplerin farklı roller üstlenmesiyle gerçekleşen bir dizi süreçtir. Yazılım Geliştirme Yaşam Döngüsü (Software Development Life Cycle - SDLC) denilen bu süreçlerde, planlama ve analiz, tasarım, uygulama (implementation), test, dağıtım ve bakım gibi aşamalar bulunur. İlgili aşamalar Şekil 1'de görüntülenmektedir. Bu aşamalarda yazılım geliştiriciler, müşteriden gelen gereksinimler doğrultusunda yazılımlarını geliştirirken, operasyonel ekipler ise ortaya çıkan yazılım ürününün dağıtımı için gerekli işlemlerin yapılması rolünü üstlenirler.

Şekil 1. Yazılım Geliştirme Yaşam Döngüsü

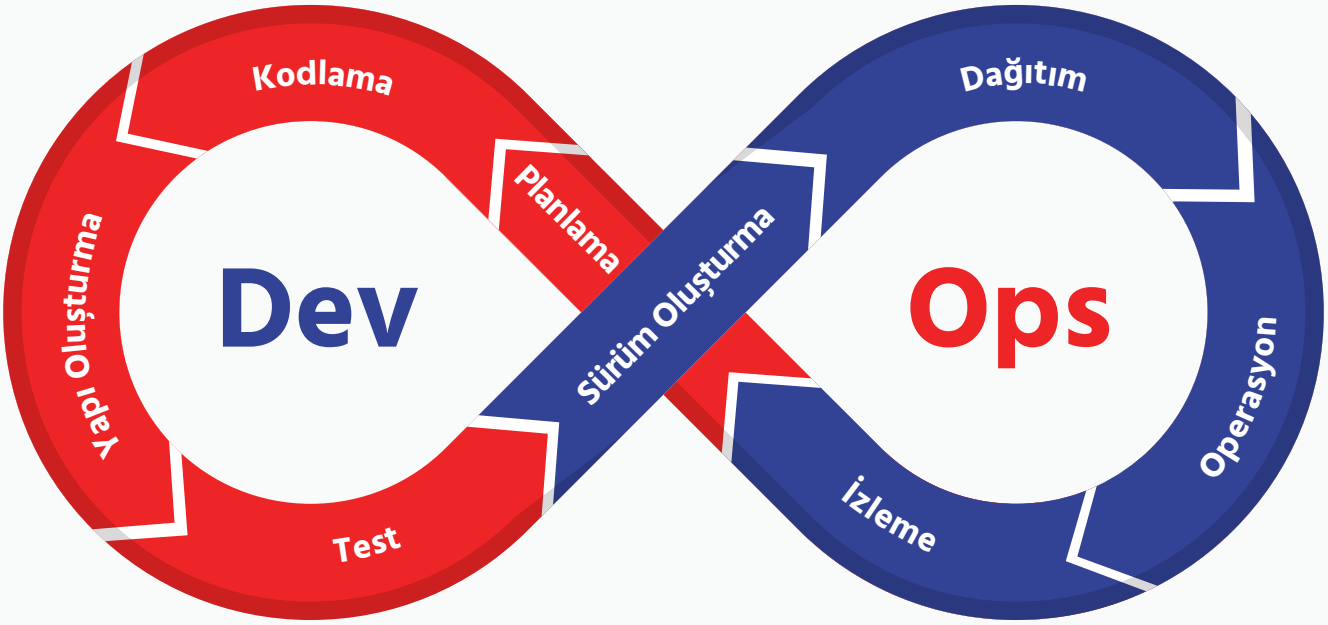
Geleneksel yazılım geliştirme süreçlerinde yazılım geliştiriciler, kodlarını kendi yerel ortamlarında geliştirirler ve operasyon ekipleriyle bu kod paylaşılarak test/üretim ortamlarına yazılımın dağıtımı yapılır. Ancak bu aşamada, bu ekipler ayrı ayrı çalışır ve geçmiş çalışmalar bu noktada bazı iletişim problemlerinin ortaya çıktığını göstermektedir (Agarwal, 2021). Geliştirme ve test/üretim ortamlarının birebir aynı olmaması, kullanılan pek çok kütüphane ve bağımlılığın (dependency) aynı şekilde test veya üretim ortamlarına aktarılamaması sonucu, kodun test/üretim ortamlarında çalışmama problemleri ortaya çıkmaktadır.

Bu problem, çevik (agile) yaklaşımın ortaya çıkmasıyla birlikte daha da önem kazanmıştır. Çevik yaklaşım, değişen müşteri isteklerine, hızlı ve kaliteli yazılım ile cevap vermeyi hedefler. 2001 yılında imzalanan çevik manifesto (Agile Manifesto, 2024) ile müşteri ihtiyaçlarının yazılımın geç fazlarında da değişebileceği ve kaliteli yazılımın sürekli teslimatı ile müşteri memnuniyetinin sağlanması gerektiği belirtilmiştir. Modern mimariler ile birlikte, organizasyonların günde yüzlerce hatta daha çok yazılım güncellemesi yapması gerekebilmektedir (Kim, Humble, Debois ve Willis, 2016). Daha hızlı dağıtım yapmak ise, yukarıda bahsedilen, kodu üretim ortamına geçirmede yaşanan problemlerin daha çok ortaya çıkmasına sebep olmaktadır.

Bu kapsamda, ortaya çıkan bu problemleri azaltmak, hızlı ve hatasız yazılım dağıtımı yapmak amacıyla Sürekli Entegrasyon (Continuous Integration - CI) ve Sürekli Teslimat (Continuous Deployment - CD) süreçleri oluşmuştur. CI/CD süreçleri, kodun yazıldığı andan itibaren üretim ortamına alınmasına kadar geçen her aşamayı kapsar. CI/CD süreçlerinde kod yazılır, ana bir repoya entegre edilir, ana repodan alınarak yapı oluşturma (build) işlemleri yapılır, gerekli testlerden geçirilir, paketlenir ve sunucuya yüklenir. Bu adımlarda en önemli amaç, tekrar eden işlemleri mümkün olduğunca otomatikleştirerek, yazılım ürününe ve müşteri isterlerine daha çok odaklanmayı sağlamaktır.

DevOps yaklaşımı, temeli çevik yaklaşıma dayanan, geliştirme (development) ve operasyon (operation) takımlarının iş birliği ve iletişimini artırarak, CI/CD süreçleri ile yazılım ürünlerinin daha verimli ve hızlı bir şekilde teslim edilmesini hedefleyen bir yaklaşım olarak 2007 yılında ortaya çıkmıştır. Çevik yaklaşımın temellerinden olan kaliteli yazılımın hızlı teslimatı konusu, 2009 yılında yapılan bir konferansta (Velocity conference) "10 Deploys per Day: Dev and Ops Cooperation at Flickr" adlı çalışmada ele alınmıştır ve çok ilgi görmüştür. Bu çalışmada, geliştirici ve operasyon ekiplerinin ortak hedefler ve iş birliği ile çalışmasıyla yazılım dağıtım süreçlerinin daha verimli ve hızlı bir şekilde yapıldığından bahsedilmiştir (Kim, Humble, Debois ve Willis, 2016).

DevOps süreçleri, yazılım geliştirme yaşam döngüsüne benzer şeklinde birbirini takip eden bir dizi aşamadan oluşur. Modern mimarilerde DevOps süreçleri planlama, kodlama, yapı oluşturulması, kodun test edilmesi, paketlenmesi, sunucuya yüklenmesi, çalıştırılması ve yazılım ürününün olası hatalara karşı izlenmesi adımlarını içerir. İlgili adımlar Şekil 2'de gösterilmiştir.



Şekil 2. DevOps Aşamaları

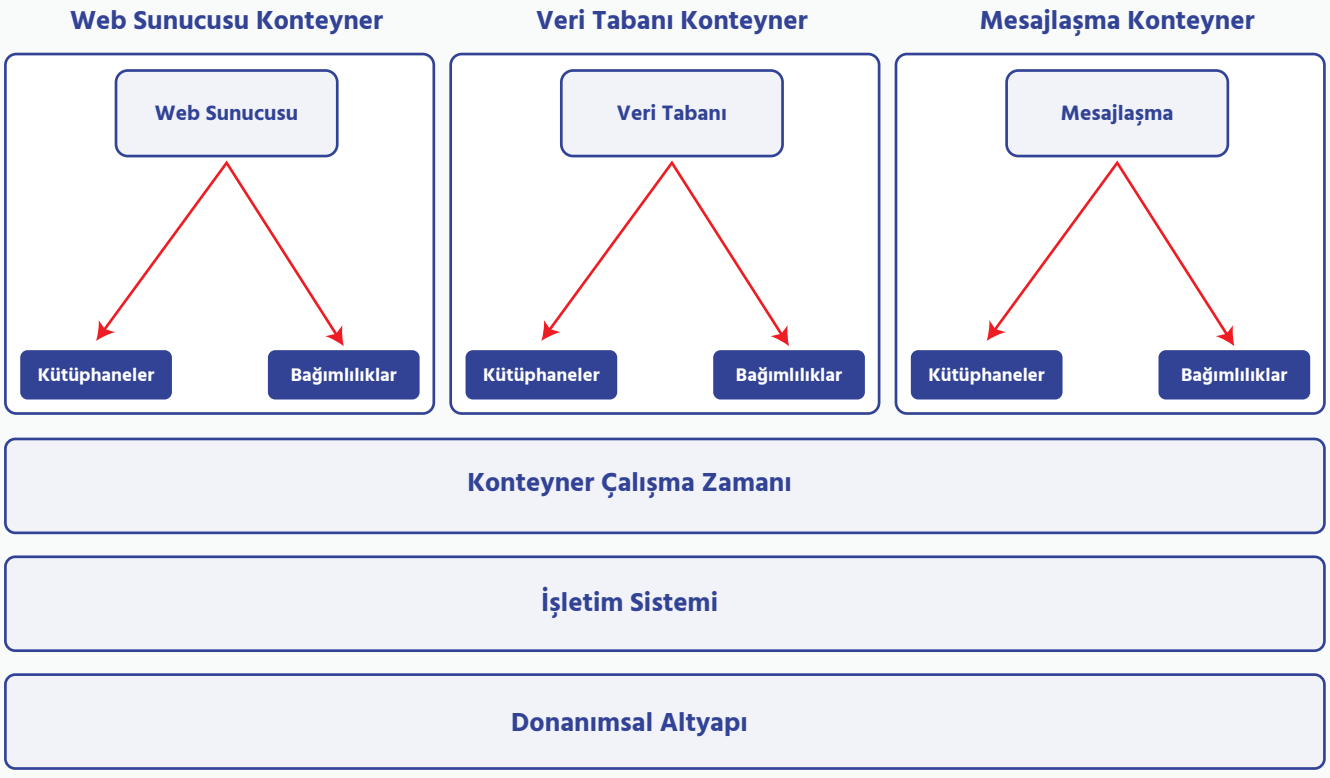
DevOps döngüsünün temelinde süreklilik bulunur. Döngü öncelikle planlama ile başlar. Analizi yapılan ve nasıl yapılacağına karar verilen müşteri gereksinimleri sonucu kodlama aşamasına geçilir. Kod yazımı sonrası yapı oluşturulması ile gerekli testler yapılır ve kod paketlenir. Dağıtımı yapılan yazılım ürününün izleme ve takibi yapılarak, olası hatalara karşı hazırlıklı olunur. Çevik yaklaşıma uygun olarak, yazılımın yeni dağıtımında olacak yeni özellikler planlanarak döngü devam eder. Böylelikle, sürekli ve hızlı teslimat sağlanmış olur.

Modern Mimariler ve Konteyner Teknolojileri

Modern yazılım mimarilerinde sıklıkla küçük, esnek ve yönetimi kolay, geliştirilmesi monolitik yapılara göre çok daha kısa süren ve tek başına çalışabilen mikroservis yapıları kullanılır. Mikroservisler kendi veri tabanları bulunan, dağıtımı ayrı yapılan, yazılımda bir hata oluştuğunda tüm sistemi bozmayacak şekilde tasarlanan yapılardır. Modern mimarilerde yazılım ürünleri pek çok mikroservis uygulamasının bir araya gelmesiyle oluşur. Mikroservisler, çalışmaları için gerekli tüm bileşenlerle birlikte konteyner içerisinde dağıtılır.

Konteyner teknolojileri, modern mimarilerde mikroservislerin yaygın kullanılmasıyla CI/CD süreçlerinin önemli bir bileşeni olmuştur. Konteynerler, uygulama kodlarını, çalışma ortamını, uygulamanın kullandığı tüm kütüphane ve bağımlılıkları bir paket haline getirir. Geleneksel sanallaştırma yöntemlerine göre daha hızlı ve hafif çalışırlar, çünkü işletim sistemi sanallaştırılmaz. Bu özellik, kaynak kullanımı ve hız konusunda avantaj sağlar.

Konteynerler içinde, uygulamanın çalışabilmesi için gerekli olan her şey bulunur. Böylelikle uygulamalar geliştirme, test ve üretim gibi tüm ortamlarda aynı şekilde çalışabilecek durumda olur. Yazılım geliştiriciler de, üretim ortamlarındaki konfigürasyonların aynısına kendi geliştirme ortamlarında sahip olur. Bu yöntem ile geliştiriciler yerel geliştirme ortamlarında, üretim ortamlarında oluşabilecek hataları görür ve geç kalmadan olası hataları çözebilirler. Yazılım geliştiriciler tarafından sıklıkla kullanılan "Benim makinemde çalışıyor" durumunun önüne geçilmiş olur (Bird, 2016). Bu aşamada, manuel süreçlerden çok otomatikleştirilmiş süreçler ile bu ortamların oluşturulması gerekir. Bunu versiyon kontrol sistemlerinde tutulan komut ve yapılandırma dosyaları ile yapmak mümkündür.



Şekil 3. Modern Mimarilerde Konteyner Teknolojisi (Agarwal, 2021)

Yazılım projelerinde, birden fazla mikroservis ve konteyner olan dinamik ortamlarda, konteynerlerin takibi zorlaşmaktadır. Bu noktada, konteyner orkestrasyon kavramı ortaya çıkar. Konteyner orkestrasyonun sağlanması ile konteynerlerin otomatik dağıtılması, yönetilmesi, izlenmesi ve ölçeklendirilmesi gibi işlemler kolay hale gelir.

Konteyner orkestrasyonu ile uygulamaların istenen durumda çalışması sağlanır. Otomatik dağıtım ve planlama yapılarak, kaynak yönetimi (CPU, bellek vb.) sağlanır. Arızalı duruma geçen konteynerlerin, yeniden başlatılması mümkün olur.

Modern Mimarilerde DevOps Aşamaları

Modern mimarilerde DevOps süreçlerinden bahsederken aşağıdaki adımlar temel olarak ele alınmaktadır.

1. Kaynak Kod Yönetimi

Kaynak kod yönetimi, genel olarak yazılım projelerinin kaynak kodunun takip edilmesi sürecidir. Kodda değişiklik ve sürüm takibini sağlamakla birlikte aynı zamanda yazılım geliştiricilerin iş birliği içinde çalışmasını sağlar. Otomatikleştirilmiş bir DevOps sürecinin en temel adımlarından biridir. Bu aşamada, sürüm kontrol sistemleri kullanılır. Kullanılabilecek en yaygın açık kaynak araç olarak Git (Git, 2024) söylenebilir. Git'te kaynak kod üzerinde dal açma, dalları birleştirme, değişiklikleri izleme ve geri alma süreçleri kolayca yönetilebilir. Açık kaynak bir araç olması sebebiyle de kaynak kod yönetiminde en çok tercih edilen araçtır.

2. Yapı Oluşturma

Yazılım geliştirme süreçlerinde yapı oluşturma aşaması, oluşturulan kaynak kodun, çalışan yazılıma dönüştürülmesi için kodun bağımlılıklarıyla birlikte çalıştırılabilir hale getirilmesi ve paketlenmesi sürecidir. Bu aşamada kodun derlenmesi, bağımlılıkların projeye dahil edilmesi, projede yazılan testlerin çalıştırılması, kodun paketlenmesi ve otomatik dağıtımın yapılması gibi adımlar bulunur. Modern mimarilerde en yaygın açık kaynak yapı oluşturma araçları Maven ve Gradle (Maven, 2024; Gradle, 2024) olarak söylenebilir.

3. Sürekli Entegrasyon / Sürekli Dağıtım (CI/CD)

Hızlı ve güvenli teslimatın yapılabilmesi için, kodda yapılan değişikliklerin hızlı ve sürekli bir şekilde merkezi bir depoya gönderilmesi Sürekli Entegrasyon (CI) adımında gerçekleşir. Bu aşamada, kod merkezi depodan çekilir, kod gerekli yapı oluşturma ve test işlemlerinden geçer. Kod, analizden geçirilir ve uygulama imajı oluşturulur. Böylece kodda daha sonra karşılaşılabilecek pek çok hata, henüz sürekli entegrasyon aşamasındayken yakalanmış olur. Sürekli Dağıtım (CD) aşamasında ise, kodun derlenmesi ve test edilmesi işlemleri sonrası test/üretim ortamına dağıtılması işlemleri yapılır. En yaygın kullanılan açık kaynak CI/CD araçları GitLab CI/CD, Jenkins, Tekton ve Travis (GitLab, 2024; Jenkins, 2024; Tekton, 2024; Travis, 2024)'tir. CD aşamasında ise en yaygın kullanılan açık kaynak araç olarak ArgoCD (ArgoCD, 2024) söylenebilir.

4. Konfigürasyon Yönetimi / Kod Olarak Altyapı (Infrastructure as Code - IaC)

DevOps süreçlerinde sunucu güncellemeleri, uygulama dağıtımları kısaca altyapısal pek çok işlemin yapılandırma dosyaları aracılığıyla otomatik ve tekrar edilebilir şekilde yapılması konfigürasyon yönetimi aşamasında yapılır. IaC'de altyapı kod olarak yönetilerek, tekrar edilebilir işlemlerin yapılandırma dosyaları aracılığıyla yapılması sağlanır. Böylece altyapısal işlemler de otomasyon ile yönetilebilir hale gelir. Yapılandırma dosyaları (YAML, JSON vb.) kod olarak tanımlandığı için, versiyon takip sistemleri ile takip edilebilir ve yönetilebilir. Ansible, Puppet, Chef ve Terraform (Ansible, 2024; Puppet, 2024; Chef, 2024; Terraform, 2024) gibi araçlar bu alanda kullanılan en yaygın açık kaynak araçlardır.

5. Konteynerleştirme

Konteynerleştirme, uygulama kodlarını tüm bağımlılıklarıyla paketleyerek, işletim sisteminden yalıtılmış bir şekilde çalışır hale getirme işlemidir. Konteynerler, uygulama kodlarını ve tüm bağımlılıkları taşınabilir hale getirir. Sanal makinelerden daha hafif yapılardır. Modern mimarilerde mikroservislerin yaygınlaşmasıyla, mikroservis kodlarının konteyner yapıları içerisinde dağıtımı sağlanır. Monolotik uygulamalar da konteynerleştirilebilir. Bu aşamada kullanılabilecek en yaygın açık kaynak araçlar Docker ve Podman (Docker, 2024; Podman, 2024)'dir.

Konteynerlerin çalışması için uygulama kod ve bağımlılıklarının paketlenmesi ile oluşturulan imajlar kullanılır. Uygulama imajları, oluşturulduktan sonra bir imaj deposunda tutulur. Docker Hub, Harbor ve Quay (Docker Hub, 2024; Harbor, 2024; Quay, 2024) en yaygın kullanılan açık kaynak imaj depolarıdır.

6. Orkestrasyon

Modern mimarilerde, birçok mikroservisin bir araya gelmesiyle oluşan yazılım ürünleri, birden fazla konteyner çalıştırılarak üretim ortamlarında sunulur. Dinamik ortamlarda sayıları gittikçe artan konteynerlerin takibi için orkestrasyon araçları kullanılmaktadır. Konteyner orkestrasyon araçları ile konteynerlerin sürekli izlenerek istenen duruma gelmesi, çöken konteynerlerin yeniden başlatılması, yük arttığında/azaldığında konteynerlerin ölçeklendirilmesi gibi işlemler yapılır. Bu aşamada kullanılabilecek en yaygın açık kaynak araçlar Kubernetes, Docker Swarm ve Apache Mesos (Kubernetes, 2024; Docker Swarm, 2024; Apache Mesos, 2024)'tur.

7. İzleme (Monitoring)

Uygulamaların üretim ortamlarına alındıktan sonra sağlıklı bir şekilde çalıştığını takip etmek ve olası hataların önüne geçebilmek için sistemlerin izlenmesi çok kritik bir konudur. Uygulama metriklerini, ağ ve altyapıyı izleme ile olası uygulama ve sistem kaynaklarının fazla tüketimi (CPU, bellek, disk vb.) gibi sorunların önüne geçmek mümkün olur. Bu aşamada, izleme araçları ile uyarı sistemleri oluşturmak mümkündür. Bu uyarılar özelleştirilerek istenilen ve belirlenen eşik değerlerinin geçilmesi durumlarında ilgili kişilere uyarı gönderimi sağlanır. En yaygın kullanılan açık kaynak izleme araçları Prometheus, Grafana ve Zabbix (Prometheus, 2024; Grafana, 2024; Zabbix, 2024) olarak söylenebilir. Bu aşamada uygulama performansı izleme de kritik rol oynar. Jaeger APM (Application Performance Monitoring) (Jaeger, 2024) gibi açık kaynak araçlarla mikroservisler arasındaki iletişim, dar boğaz gibi durumlarda sorunların tespiti yapılabilir.

DevOps süreçlerinde yukarıdaki aşamalara ek olarak bulut (cloud) teknolojileri de kritik bir rol oynar. Uygulama altyapısı için kurulacak kümeler (clusters), organizasyonların kendi sunucularında (bare-metal) kurulabileceği gibi bulut sağlayıcılar tarafından sağlanan kaynaklar üzerinde de kurulabilir. Organizasyonlar, ücretli olarak sağlanan bu sanal makineler üzerinde kümelerini kurarak, bulut sağ-

layıcıların sağladığı altyapı, kurulum ve konfigürasyon kolaylıklarından faydalanabilirler. Bulut sağlayıcılar üzerinde CI/CD adımları da çalıştırılabilir. Bulut sağlayıcılara örnek olarak Amazon Web Services, Microsoft Azure, Google Cloud, IBM Cloud, VMWare Cloud verilebilir.

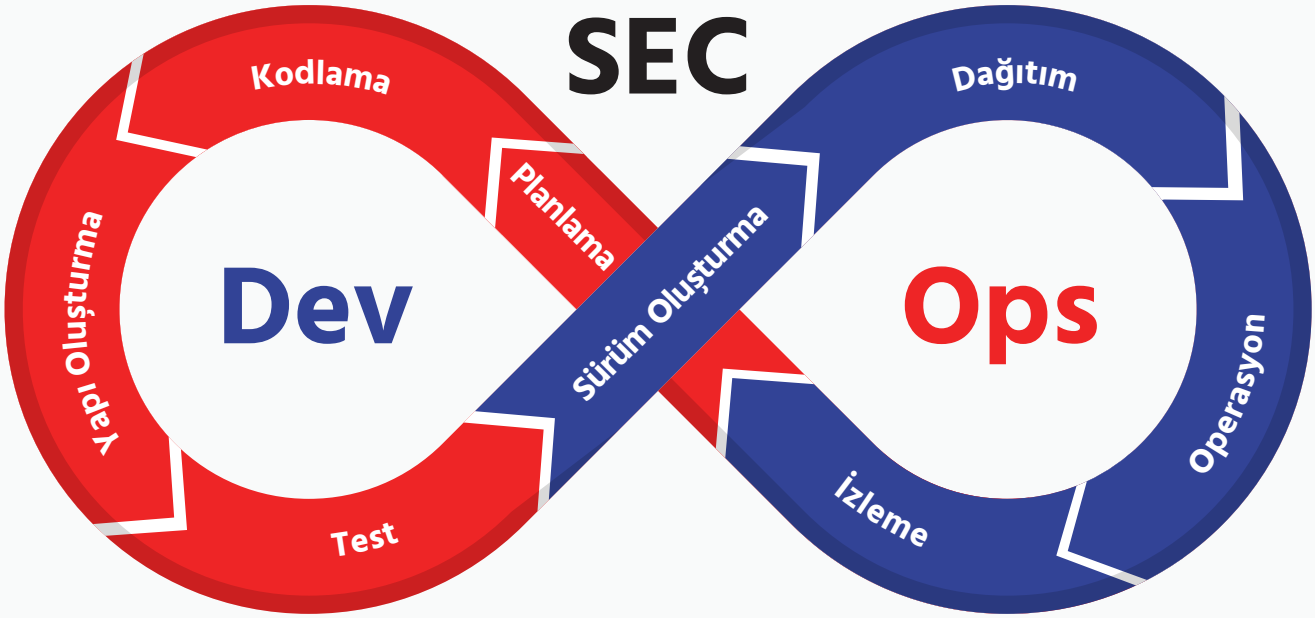
Yukarıdaki adımları çok iyi işleten bir DevOps süreci oluşturulduğunda, kaynak koddan başlayarak, uygulama kodu kod deposundan çekilir, yapı oluşturulur, CI/CD adımlarında testler çalıştırılır, imaj oluşturulur ve uygulama imajı konteyner halinde çalıştırılarak, bir konteyner orkestrasyon aracıyla uygulama yönetilir ve sonrasında sürekli izlemesi sağlanır. Ancak çalışan bir yazılım ürününün, güvenliğinden emin olmak gereklidir. DevOps süreçleri bize hız, ölçeklenebilirlik ve pek çok fonksiyonalite sağlamasına rağmen, güvenlik açıklarına karşı korunması gereken pek çok durum gerektirir. Mikroservislerin de kullanılmasıyla birlikte, yazılım ürünlerinde saldırılara açık yüzey alanları daha da genişler. Ancak, görüldüğü üzere yukarıdaki adımlarda güvenlikten bahsedilmemektedir.

Geleneksel yazılım süreçlerinde güvenlik genellikle en sona bırakılan konudur. Oluşturulan yazılım ürünü güvenlik ekiplerine gönderilir. Güvenlik ekipleri yazılımı inceleyerek, olası korunmasız, tarihi geçmiş üçüncü parti kütüphaneleri, lisanslama meselelerini, hassas verilerin veya korunmasız imajların olup olmadığını ve yanlış konfigürasyonlarının geçilip geçilmediğini test ederler ve sızma testi gibi güvenlik testlerini yaparlar. Bu süreçler sonrasında, çıkan hata veya eksiklikler tekrar yazılım ekiplerine bildirilir ve DevOps boru hattı (pipeline) tekrar başa dönmüş olur. Tekrar eden bu adımlar, yazılım ürününün erken teslimatını geciktirir. Bu noktada DevSecOps süreçleri, yazılım dünyasında önemli bir rolü üstlenir ve güvenlik kavramını DevOps süreçlerine entegre eder.

DevSecOps Nedir?

Geleneksel yazılım mimarilerinde, güvenlik katmanlarının yazılım geliştirmesi yapıldıktan sonra eklenebileceği düşünülmektedir çünkü ana amaç uygulama geliştirmeyi tamamlamaktır. Ancak, bu yaklaşımda geç fazlarda bulunan güvenlik açıkları çok daha maliyeti yüksek sonuçlara yol açmaktadır (Sehgal, 2023). DevOps aşamalarını incelediğimizde, güvenlikten bahseden herhangi bir katman olmadığı görülmektedir. Otomatikleştirilmiş DevOps süreçleri sonrasında, manuel güvenlik testlerinin yapılması ise yazılım teslimatını geciktirmektedir. Bir yazılımın güvenli geliştirme ve dağıtımı için güvenliğin tek bir adımda yapılan bir işlem olmaması gerekir. Güvenlik, en sola yani en başa kaydırılarak SDLC'nin her adımında yer alması gerekmektedir.

DevSecOps (Development-Security-Operations), DevOps süreçlerine baştan sona kadar her bir adıma güvenliğin eklenmesi ile gerçekleşmektedir. Kod yazma aşamasından başlayarak güvenli ve kaliteli kod yazılması, uygulamanın kullandığı paketlerin güvenlik kontrolü, imajın güvenilir bir şekilde oluşturulması, uygulamanın güvenilir dağıtımının yapılması gibi pek çok aşamaya güvenlik dahil edilir. Böylece sonradan çıkabilecek güvenlik riskleri en aza indirilir ve otomatikleştirilmiş güvenlik süreçleri ile hızlı ve verimli güvenlik sağlanmış olur. Şekil 4'te DevSecOps aşamaları gösterilmiştir.



Şekil 4. DevSecOps Aşamaları

DevOps pratiklerinin uygulanması gibi DevSecOps pratiklerinin organizasyon içerisinde uygulanması da organizasyonun DevSecOps kültürünü benimsemesi ile olur (Suehring, 2024). Organizasyonlarda yazılım ekipleri, operasyonel ekipler ve güvenlik ekipleri iş birliği içinde çalışmalıdır. Güvenlik ekipleri, her aşamada güvenlik konusunda güvenlik kontrollerine devam eder. Güvenlik kurallarını tanımlayarak, tüm sürecin güvenliğini belirler. Yazılım ve operasyon ekipleri ise, güvenliğin herkesin sorumluluğu olduğunu bilerek ve her aşamada güvenliği hesaba katarak uygulama geliştirme süreçlerini ve operasyonel süreçleri yürütürler.

Yazılım dünyasında, güvenlik aşamasından bahsederken hem yazılım geliştiriciler, hem operasyonel ekipler hem de güvenlik ekipleri tarafından gerçekleştirilebilecek en önemli adımlardan biri İnternet Güvenliği Merkezi'nin (Center for Internet Security - CIS) hazırlamış olduğu dokümanların (CIS Benchmarks) incelenmesidir. Bu dokümanlar, alanında uzman kişiler tarafından pek çok araç için ayrıntılı bir şekilde hazırlanmıştır ve kullanılacak araçlar için, uygulanması gereken güvenlik uygulamalarını listeler. PostgreSQL, MariaDB gibi veri tabanları, farklı Linux dağıtımları, bulut sağlayıcılar, Kubernetes, Docker gibi DevOps araçları ve daha pek çok araca ait güvenlik adımlarını listeleyen çok sayıda CIS dokümanı bulunmaktadır (CIS, 2024). Örneğin PostgreSQL gibi bir veri tabanı ya da Kubernetes gibi bir orkestrasyon aracı kullanacağımız zaman, yönetim ayrıcalıklarını ve izinleri kısıtlama, kullanılmayacak bağlantı noktalarının kapatılması gibi işlemlerin yapılması gerekir. Bu gibi kontrol edilmesi gereken noktalar CIS dokümanlarında yer almaktadır. Bu nedenle, DevSecOps süreçlerinde kullanılacak araçlar için CIS dokümanlarının kontrol edilmesi önemli bir adımdır.

Konteyner Güvenliği

Modern mimarilerde mikroservislerin dağıtımı için en iyi seçenek olan konteyner teknolojisini kullanırken, konteynerin güvenliğini sağlamak çok kritiktir. Konteynerlerin çalışması için tanımlanması gereken kullanıcı yetkileri, güvenlik açıklarına yol açabilir. Kullanıcı izinleri, imaj kullanımı ve ağ güvenliği hakkında dikkat edilmesi gereken en önemli konular aşağıdaki şekilde sıralanabilir.

1. **İzinler:** Konteynerin çalışma zamanında sahip olacağı izinlerdir. Konteynerlerin çalışabilmesi için, sistem üzerinde belirli kaynaklarda işlem yapabilecek yetkilere sahip olması gerekir. Ancak gereğinden fazla izin vermek, sıklıkla yapılan hatalar arasında yer alır. Burada en sık karşılaşılan durum konteynerlerin "root" kullanıcısı ile çalıştırılmasıdır. "Root" kullanıcısı, sistem kaynakları üzerinde tüm işlemleri yapmaya yetkilidir. Bir konteynere bu yetkiyi vermek, konteynerin sistem üzerinde gereğinden fazla yetkili olmasına sebep olur. Konteyner içine girmeyi başaran bir saldırgan, sistem üzerinde çok fazla işlem yapabilir hale gelecektir. Çözüm olarak, kısıtlı yetkilerle konteynerleri çalıştırabilecek özel kullanıcılar yaratılmalı ve konteynerler bu kullanıcılar ile çalıştırılmalıdır.
2. **İmajlar:** Bir uygulamanın konteyner içerisinde çalışması için, hem uygulama kodu hem de uygulama dışı web sunucusu, veri tabanı gibi imajlar kullanılır. Bu noktada, dikkat edilmesi gereken adımlar bulunmaktadır.
   a. **Güvenilir Kaynaklardan İmaj Kullanımı:** Konteyner içerisinde kullanılacak her bir imajın güvenliği, konteyner güvenliğini de etkiler. Docker Hub ve farklı imaj depolarında bireysel oluşturulmuş pek çok imaj bulunmaktadır. Bu depolarda güvenlik taramalarından geçmiş resmi imajların kullanımı kritik öneme sahiptir.
   b. **Güncel İmajların Kullanımı:** Resmi imajların güncel versiyonları kullanılmalıdır. Özellikle güvenlik açığı bulunan, güvenlik güncellemeleri gelmiş imajların eski versiyonlarının kullanılmamasına dikkat edilmelidir. Ek olarak, boyutu küçük imajların seçimi saldırı yüzeyini azaltmak için önemlidir.
   c. **İmaj İmzalama:** Konteyner imajlarının değiştirilmemesi ve bütünlüğünün sağlanması için imaj imzalama yöntemi kullanılır. İmaj imzalama yöntemi ile imajın kaynağı da doğrulanmış olur. İmaj imzalamak için Notary ve Cosign (Notary, 2024; Cosign, 2024) gibi açık kaynak araçlar kullanılabilir.
   d. **İmaj Tarama:** Konteyner imajları oluşturulduktan sonra, bu imajların güvenlik analizi için imaj tarama işlemi yapılır. İmajlardaki güvenlik açıkları tespit edilir. Açık kaynak araçlara örnek olarak Clair ve Trivy (Clair, 2024; Trivy, 2024) verilebilir.

3. **Ağ Güvenliği:** Konteynerlerin çalıştığı ağlarda, konteynerler arası güvenli iletişimin sağlanması için ağ güvenliğini sağlamak gereklidir. Kubernetes gibi konteyner orkestrasyon araçlarında, ağ politikalarıyla kuralları belirlemek ve iletişim izinlerini kısıtlamak mümkündür. Ayrıca RBAC (Role-Based Access Control – Rol Tabanlı Erişim Kontrolü) ile gerekli izinlerin verilmesi sağlanır. Mikroservisler arası iletişimi sağlayan servis ağı (service mesh) araçları da güvenlik katmanları içerir. Bu amaçla Istio ve Linkerd (Istio, 2024; Linkerd, 2024) gibi açık kaynak araçlar kullanılabilir.
4. **Kaynak İzolasyonu:** Konteynerlerin aşırı kaynak tüketimini engellemek ve konteynerlerin kullanacağı CPU, bellek gibi kaynakların kullanımını kısıtlamak gerekir. Kaynak izolasyonu projeleri ayırma (namespace ile), Kubernetes politikaları ve kontrol grupları (cgroups) ayarları ile sağlanabilir.

Modern Mimarilerde DevSecOps Aşamaları

Modern mimarilerde DevSecOps süreçlerinden bahsederken aşağıdaki adımlar temel olarak ele alınmaktadır.

1. Güvenli Kod Gözden Geçirme

Güvenli yazılım ürünü ortaya çıkarmak, güvenli kod yazma ile başlar. Kod yazma aşamasında oluşabilecek SQL Injection (veri tabanı sorgularına SQL sızıntısı ile erişme), Cross-Site Scripting (XSS) (Javascript gibi betik dilleri ile tarayıcı üzerinden kod çalıştırma), Path Traversal (dosya yolları üzerinden başka dosyalara erişebilme) gibi kod düzeyinde yapılan güvenlik hataları ile henüz kod yazma aşamasında saldırılara açık yüzeyler oluşturulmuş olur. Güvenli kod gözden geçirme aşaması ile koddaki hatalar, güvenlik açıkları ve koddaki kötü kokular (code smells) erken aşamalarda bulunur. Pek çok Entegre Geliştirme Ortamı'na (Integrated Development Environment - IDE) uyumlu olarak çalışabilen ve kod yazım aşamasında bu tür hataları gösteren açık kaynak araçlar bulunur. Bu aşamada en yaygın kullanılan açık kaynak araç olarak SonarQube (SonarQube, 2024) söylenebilir.

2. Statik Uygulama Güvenlik Testi (Static Application Security Testing - SAST)

Statik uygulama güvenlik testi, uygulamanın kaynak kodunu veya ikili (binary) dosyalarını analiz ederek güvenlik zafiyetlerini bulmaya çalışan aşamadır. Analizler, kod çalıştırılmadan, statik kod üzerinden yapılır. Genellikle uygulamanın CI/CD boru hattı çalıştırılırken, statik kod analiz araçları ile SAST aşaması gerçekleştirilir. SAST araçları, CI/CD araçlarıyla entegre çalışabilir. SAST işlemi sonrası kod kapsama (code coverage), zafiyet oranı ve hataları onarma zamanı (remediation time) gibi metrikler bir rapor halinde sunulur. Bu aşamada en yaygın kullanılan açık kaynak araç olarak SonarQube (SonarQube, 2024) söylenebilir.

3. Dinamik Uygulama Güvenlik Testi (Dynamic Application Security Testing - DAST)

Statik uygulama testleri uygulama çalışmazken uygulanırken, dinamik uygulama güvenlik testleri ise uygulama çalışırken uygulanarak güvenlik açıklarını tespit eder. DAST araçları, kasıtlı olarak uygulamaya bozuk girdiler (input) göndererek uygulamanın nasıl tepki vereceğini test eder. Girdi alanlarına uygun olmayan formatta verilerin gönderilmesi, yetkilendirme mekanizmalarının geçilmeye çalışılması, hassas verilere erişilmeye çalışılması gibi testler yapılır. Test ortamında yapılan bu testler, üretim ortamında olabilecek hataları önceden yakalayabilir. DAST araçları CI/CD aşamalarına entegre edilebilir. OWASP (Open Web Application Security Project) Zed Attack Proxy ve Nuclei (ZAP, 2024; Nuclei, 2024), en yaygın kullanılan açık kaynak DAST araçlarına örnek olarak verilebilir.

4. Yazılım Kompozisyon Analizi (Software Composition Analysis – SCA)

Bir yazılım ürünü ortaya çıkarırken, yazılıma kaynak kodlara ek olarak pek çok üçüncü parti kütüphane adı verilen yazılım ilave edilir. Daha önce başkaları tarafından yazılmış bu kütüphaneler ile aynı iş tekrar tekrar yapılmamış olur. Ancak bu aşamada başka kütüphanelerin güvenilirlik konusu ön plana çıkar. Yazılım kompozisyon analizi, kaynak koda ilave olarak kullanılan ve bağımlılık adı verilen bu kütüphanelerin güvenliğini ele alır. SCA ile bu bağımlılıkların güncelliği, zafiyetleri ve varsa lisans uyumsuzlukları saptanır. SCA araçları, Ulusal Standartlar ve Teknoloji Enstitüsü (National Institute of Standards and Technology - NIST) tarafından yayınlanan güvenlik açıklarını kontrol eden National Vulnerability Database (NVD, 2024) gibi veri tabanlarındaki güvenlik açıklarını takip eder. Bu araçlar CI/CD araçlarıyla entegre çalışabilir. SCA araçları ile yazılımda bulunan toplam bağımlılık sayısı, zafiyetli bağımlılık sayısı, güncel olmayan bağımlılıklar, lisansı uygun olmayan bağımlılıklar, tekrar eden bağımlılıklar gibi metrikler elde edilebilir. Bu alanda en yaygın kullanılan SCA araçlarına örnek olarak OWASP Dependency-Check (Dependency-Check , 2024) verilebilir.

5. Gizlilik Yönetimi (Secret Management)

DevOps sürecinde kullanılacak parolalar, şifreler, sertifikalar, API anahtarları gibi hassas bilgilerin güvenli bir şekilde tutulması gerekir. Bu hassas bilgiler düz metin olarak saklanmamalı ve kimlerin bu bilgilere erişebileceği konusunda kısıtlamalar getirilmelidir. Gizlilik yönetimi araçları, bu bilgileri ortak bir noktadan yöneterek erişim kontrolleri konusunda tanımlamalar yapmayı sağlar. Açık kaynak gizlilik yönetimi araçlarına örnek olarak HashiCorp Vault (Vault, 2024) verilebilir.

6. Orkestrasyon Bileşenlerinin Denetlenmesi

DevSecOps süreçlerinde konteyner orkestrasyonu için kullanılan araçların denetlenmesi de kritik bir adımdır. Orkestrasyon için en sık kullanılan araçlardan olan Kubernetes için güvenliğin kontrol edilmesi gerekir. Kube-bench (Kube-bench, 2024), CIS dokümanlarına göre Kubernetes'in güvenli

bir şekilde kullanımı için, kube-hunter (Kube-hunter, 2024) ise Kubernetes'teki güvenlik zafiyetlerini tespit etmek için kullanılan araçlara örnek olarak verilebilir.

Yukarıdaki adımların DevOps süreçlerine entegre edilmesiyle, güvenlik katmanı kod yazımından itibaren DevOps süreçlerine entegre edilmiş olur. Uygulamanın üretim ortamına aktarılmasından sonra, sürekli izleme ile uygulamaların ve kümelerin takibi yapılarak, olası hataların önüne geçilir.

DevOps/DevSecOps Süreçlerinin Avantajları

DevOps süreçleri, CI/CD süreçleri ve bu süreçlerde kullanılan otomasyon araçları ile hızlı ve hatasız yazılım teslimatının yapılmasını sağlar, böylelikle yazılım yaşam döngüsü de hızlanmış olur. Yazılım, son kullanıcıya ulaşmadan önce gerekli testlerden geçirilir. Müşterilerin hızlı bir şekilde çözülmesi gereken ihtiyaçları, kolay ve hızlı bir şekilde yazılıma entegre edilebilir. Yazılım geliştirici ve operasyon ekiplerinin ortak bir dil ve amaç ile çalışması ile verimlilik artar. Üretim ortamında yakalanan hataların maliyeti çok yüksektir. CI/CD süreçleri ile hata oranları düşer ve dolayısıyla maliyet de düşmüş olur.

DevSecOps süreçlerinde ise güvenlik katmanı ön plandadır. DevSecOps süreçlerinde uygulanan adımlar ve kullanılan araçlar, güvenlik açıklarının erken tespitini sağlar. Modern mikroservislerin getirdiği saldırı yüzeyinin artması sonucu ortaya çıkabilecek güvenlik açıklarına karşı, yazılım geliş-

tirmenin ve dağıtımın her aşamasında önlem alınmış olur. CI/CD süreçlerine entegre edilen kod kalite analizi ile kodun kalitesi artar. Kaliteli kod üretimi, hatasız yazılım için çok kritiktir.

DevOps/DevSecOps süreçlerinde otomasyonun sağlanması ile manuel pek çok adım ortadan kalkar ve bu aşamalarda yaşanabilecek insan hatalarının önüne geçilmiş olur. Müşteri ihtiyaçlarının hızlı karşılanması ve güvenilir yazılım teslimatı ile müşteri memnuniyeti artar. Böylelikle çevik yaklaşımda yer alan temel hedeflere de ulaşılmış olur. Tüm bu avantajlar doğrultusunda, DevOps/DevSecOps süreçleri, modern mimarilerde uygulanması gereken iyi bir pratiktir.

DevOps/DevSecOps Süreçlerinde Yaşanan Zorluklar

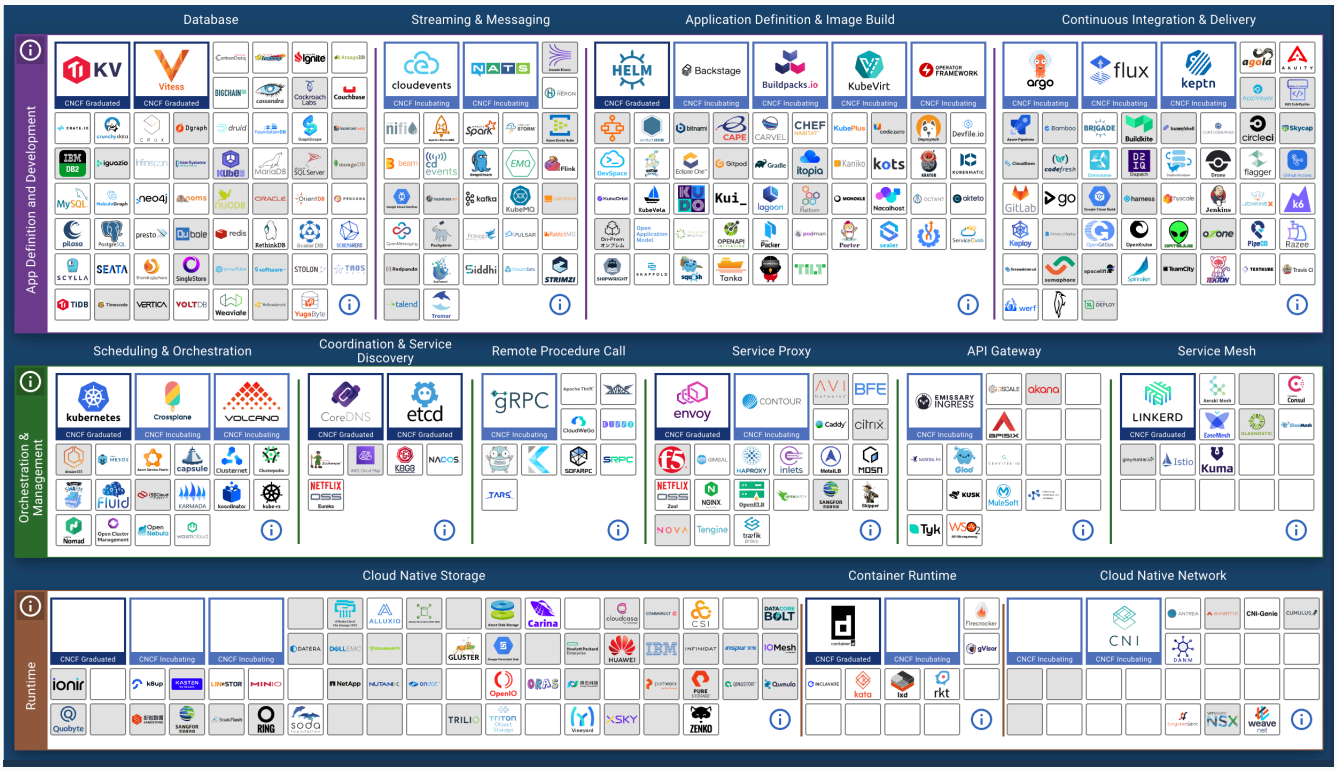
DevOps/DevSecOps süreçlerine geçişin en önemli adımı, organizasyonun ve ilgili tüm ekiplerin kültürel değişimi kabul etmesidir. Geleneksel mimari ile ayrı ekipler halinde çalışmaya alışmış geliştirme, operasyon ve güvenlik ekipleri, alışmış oldukları düzen ile devam etmek isteyebilir ve yeni kültüre karşı direnç gösterebilirler. Bu noktada, organizasyon tarafından ilgili ekiplere bilinç kazandırılmalıdır ve gerekli eğitimler verilmelidir.

DevOps/DevSecOps süreçlerinde bir diğer zorluk ise kullanılan araçların seçimidir. Yukarıda bahsedilen her bir aşamada kullanılabilecek çok fazla araç bulunmaktadır. Açık kaynak araç seçiminde en önemli nokta güvenilir topluluklar tarafından geliştirilen, desteği fazla araçların seçilmesidir. Bu aşamada, Cloud Native Computing Foundation (CNCF)'ın desteklediği CNCF Landscape araçlarını seçmek iyi bir seçim olabilir (CNCF Landscape, 2024). CNCF Landscape, Şekil 5'te sunulmuştur.

Bu süreçlerde diğer bir zorluk ise, araç seçimi sonrası ilgili takımların bu araçlar hakkında eğitilmesidir. Her bir araç üzerinde uzmanlaşmak zaman alacaktır. DevOps süreci geçişlerinde bu öğrenme eğrisi hesaba katılarak proje planlamaları yapılmalıdır.

DevOps/DevSecOps süreçleri, sürekli güncelleme gerektirir. Kullanılan araçların güncellemeleri takip edilmeli, güvenlik açığı çıkan yazılımlar mutlaka kısa süre içerisinde güncellenmelidir. Güvenlik açığı çıkan bir araçta bu açığın yayınlanması ile, bu açık herkes tarafından görünür olur ve bu yazılımları kullanan uygulamalar da saldırılara açık hale gelir. Düzenli ve kritik güncellemelerle, DevOps/DevSecOps aşamalarında tüm araçlar güncel olmalıdır.

DevOps/DevSecOps aşamalarında yer alan sürekli izleme/takip de pek çok avantajı beraberinde getirmesine rağmen, ilgili ekipler tarafından ek iş yükü olarak görünebilir. İzleme/takip önem ve avantajları konusunda ilgili ekiplere gerekli eğitimler verilerek, bu konuda gerekli bilinç kazandırılmalıdır.



Şekil 5. CNCF Landscape

DevOps/DevSecOps Süreçlerine Dönüşüm Örneği

Bu bölümde, önceki bölümlerde bahsedilen aşama ve örnek araçlarla örnek bir akış oluşturulması hedeflenmiştir. Aşağıdaki akışta, her bir aşama için örnek bir araç seçilmiştir.

Öncelikle kaynak kodun yazılması ile başlayan süreçte kodlar merkezi bir kod deposuna gönderilir ve Git ile versiyon kontrolü sağlanır. Kaynak kod yazımı aşamasında, kullanılan IDE'ye SonarQube statik kod analizi aracı eklentisi eklenerek, kod yazım aşamasında kodda oluşabilecek hatalar hakkında bilgi sahibi olunur. Kullanılacak tüm araçlar için CIS Benchmark dokümanlarından faydalanılarak, olası güvenlik açıkları önceden saptanır.

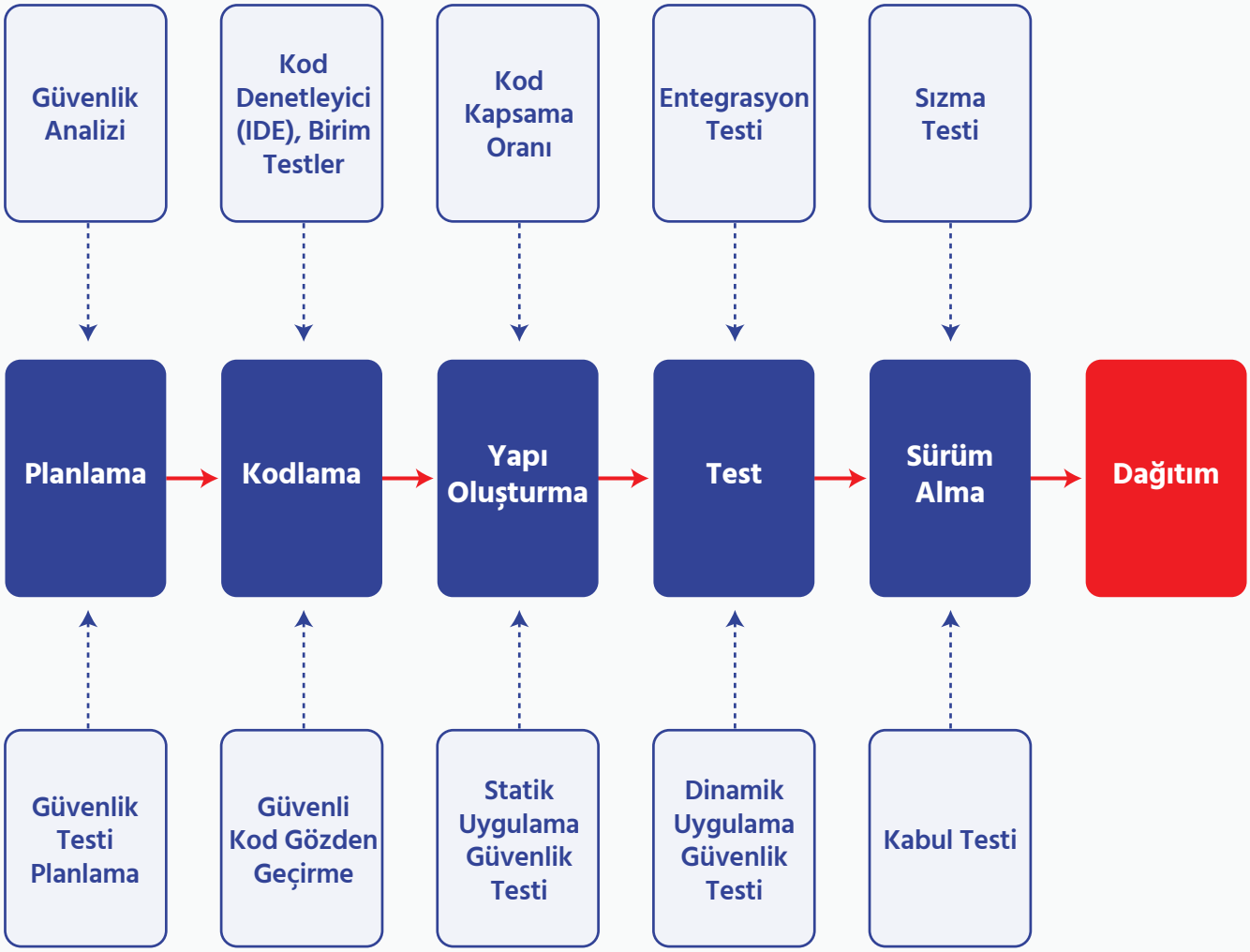
Kodun yazılması sonrası CI/CD süreci başlar. Kod, Git ile merkezi depoya gönderilir. GitLab aracı bu noktada merkezi depo olarak kullanılabilir. Burada ana dal ile birleştirilen kodlar GitLab CI/CD araçları tarafından çekilerek süreç başlatılır. Çekilen kodların yapı oluşturma (Maven), test, statik kod analizi, SCA gibi işlemleri bu süreçte gerçekleştirilir. Uygulama kodunun ilgili yapı oluşturma ve test aşamalarından güvenli geçmesi sonrası, uygulama imajı oluşturulur.

Oluşturulan uygulama imajı Quay imaj deposuna gönderilir. Burada imaj tarama işlemleri yapılır. Quay imaj deposu ile entegre çalışan Clair ile imaj taraması yapılır ve varsa zafiyetler bulunur. Güvenlik taramalarından geçmiş imaj için Cosign aracı ile imaj imzalama yapılır.

Güvenle oluşturulmuş uygulama kodu sonrası, statik kod analizi yapılmış ve güvenlik taramalarından başarıyla geçerek oluşturulmuş uygulama imajı ile Kubernetes orkestrasyon ortamında uygulama çalıştırılır. Kubernetes ortamı bulut platform veya organizasyon sunucularında kurulabilir. Yapılması gereken altyapısal konfigürasyonlar Ansible aracı ile yapılabilir. Bu aşamada, Kubernetes ortamında güvenle tutulması gereken hassas bilgiler Vault aracı ile yönetilir. Kubernetes ortamında gerekli ayarlar yapılarak, uygulamanın kaç replikasının olacağı gibi yapılandırmalar yapılır. Konteyner oluşturma sırasında dikkat edilmesi gereken adımlar göz önünde bulundurulur (Root ile konteyner çalıştırmama vb.). Test ortamında çalıştırılan uygulama ile DAST testlerine geçilir.

Sürekli teslimat (CD) aşamasında uygulamanın yönetimini sağlamak amacıyla açık kaynak ArgoCD aracı kullanılabilir. ArgoCD, Kubernetes kümelerindeki uygulamaları manifestler aracılığıyla yönetmeyi sağlar. ArgoCD sayesinde, uygulamalar güvenilir ve tekrarlanabilir bir şekilde dağıtılabilir.

Uygulamanın ilgili testlerden geçmesi ve çalıştırılması sonrası izleme ve takip süreci başlar. Prometheus ile toplanan uygulama verileri Grafana panoları oluşturularak izlenir. Bu aşamada hazır Grafana panoları kullanılabileceği gibi özelleştirilmiş panolar hazırlamak da mümkündür. Yazılımdaki mikroservisler arası iletişim ve dar boğaz gibi durumlar Jaeger APM aracılığı ile görüntülenebilir.



Şekil 6. DevSecOps Boru Hattı

Sonuç ve Öneriler

DevOps süreçleri ve araçları ile modern yazılım mimarilerinde yazılımın hızlı ve sürekli teslimatı sağlanır. DevSecOps süreçlerinde ise güvenlik en başa kaydırılarak her bir DevOps adımının güvenliği sağlanmış olur. DevOps süreçlerinin her bir adımında kullanılabilecek çok fazla araç oluşu, süreci karmaşık ve zor bir hale getirebilir. Bu aşamalarda, güvenli ve topluluk desteği fazla açık kaynak araçlardan yararlanmak iyi bir pratiktir.

DevOps/DevSecOps süreçleri, her şeyden önce bir kültürdür. Her yazılım geliştirici, geliştirdiği yazılımın güvenliğinden ve tüm bağımlılıklarıyla birlikte üretim ortamında çalışabilecek hale gelmesinden sorumlu olduğunu bilmelidir. Yazılım geliştirici, operasyon ve güvenlik takımları, iş birliği içinde çalışmalı ve ortak bir dil üzerinde anlaşabilmelidir. Ayrıca, üretim ortamına alınan yazılım ürünlerinin sürekli izleme ve takibi yapılarak, sağlıklı bir şekilde çalışmaya devam ettiği ilgili ekiplerce kontrol edilmelidir. Sonuç olarak, DevSecOps süreçleri yazılım ürünlerinin hızlı ve güvenli teslimatı için modern mimarilerde en iyi seçenektir.

Kaynakça

Agarwal, G. (2021). Modern DevOps Practices: Implement and secure DevOps in the public cloud with cutting-edge tools, tips, tricks, and techniques. Packt Publishing Ltd.

Agile Manifesto. (Haziran, 2024). https://agilemanifesto.org/principles.html

Ansible. (Haziran, 2024). https://www.ansible.com/

Apache Mesos. (Haziran, 2024). https://mesos.apache.org/

ArgoCD. (Haziran, 2024). https://argo-cd.readthedocs.io/en/stable/

Bird, J. (2016). DevOpsSec, O'Reilly Media, Inc.

Chef. (Haziran, 2024). https://www.chef.io/

CIS. (Haziran, 2024). https://www.cisecurity.org/cis-benchmarks/

Clair. (Haziran, 2024). https://github.com/quay/clair

CNCF Landscape. (Haziran, 2024). https://landscape.cncf.io/

Cosign. (Haziran, 2024). https://github.com/sigstore/cosign

Dependency-Check. (Haziran, 2024). https://github.com/jeremylong/DependencyCheck

Docker. (Haziran, 2024). https://www.docker.com/

Docker Hub. (Haziran, 2024). https://hub.docker.com/

Docker Swarm. (Haziran, 2024). https://docs.docker.com/engine/swarm/

Git. (Haziran, 2024). https://git-scm.com/

GitLab. (Haziran, 2024). https://about.gitlab.com/

Gradle. (Haziran, 2024). https://gradle.org/

Grafana. (Haziran, 2024). https://grafana.com/

Harbor. (Haziran, 2024). https://goharbor.io/

Istio. (Haziran, 2024). https://istio.io/latest/about/service-mesh/

Jaeger. (Haziran, 2024). https://www.jaegertracing.io/

Jenkins. (Haziran, 2024). https://www.jenkins.io/

Kim G., Humble J., Debois P. & Willis J. (2016). The DevOps Handbook, IT Revolution Press

Kubernetes. (Haziran, 2024). https://kubernetes.io/

Kube-bench. (Haziran, 2024). https://aquasecurity.github.io/kube-bench/dev/

Kube-hunter. (Haziran, 2024). https://aquasecurity.github.io/kube-hunter/

Linkerd. (Haziran, 2024). https://linkerd.io/

Maven. (Haziran, 2024). https://maven.apache.org/

Notary. (Haziran, 2024). https://github.com/notaryproject/notary

Nuclei. (Haziran, 2024). https://github.com/projectdiscovery/nuclei

NVD. (Haziran, 2024). https://nvd.nist.gov/

Podman. (Haziran, 2024). https://podman.io/

Prometheus. (Haziran, 2024). https://prometheus.io/

Puppet. (Haziran, 2024). https://www.puppet.com/community/open-source

Quay. (Haziran, 2024). https://quay.io/

Sehgal V. V. (2023). Implementing DevSecOps Practices, Packt Publishing Ltd.

SonarQube. (Haziran, 2024). https://www.sonarsource.com/

Suehring S. (2024). Learning DevSecOps, O'Reilly Media, Inc.

Tekton. (Haziran, 2024). https://tekton.dev/

Terraform. (Haziran, 2024). https://www.terraform.io/

Travis. (Haziran, 2024). https://www.travis-ci.com/

Trivy. (Haziran, 2024). https://trivy.dev/

Vault. (Haziran, 2024). https://github.com/hashicorp/vault

Zabbix. (Haziran, 2024). https://www.zabbix.com/

ZAP. (Haziran, 2024). https://www.zaproxy.org/

T.C. SANAYİ VE
TEKNOLOJİ BAKANLIĞI

#MİLLİ TEKNOLOJİ HAMLESİ

TÜBİTAK

İşçi Blokları Mahallesi Muhsin Yazıcıoğlu Caddesi No:51/C 06530 Çankaya/ANKARA

+90 (312) 289 92 22 - yte.bilgi@tubitak.gov.tr

TÜBİTAK - BİLGEM Yazılım Teknolojileri Araştırma Enstitüsü (YTE)