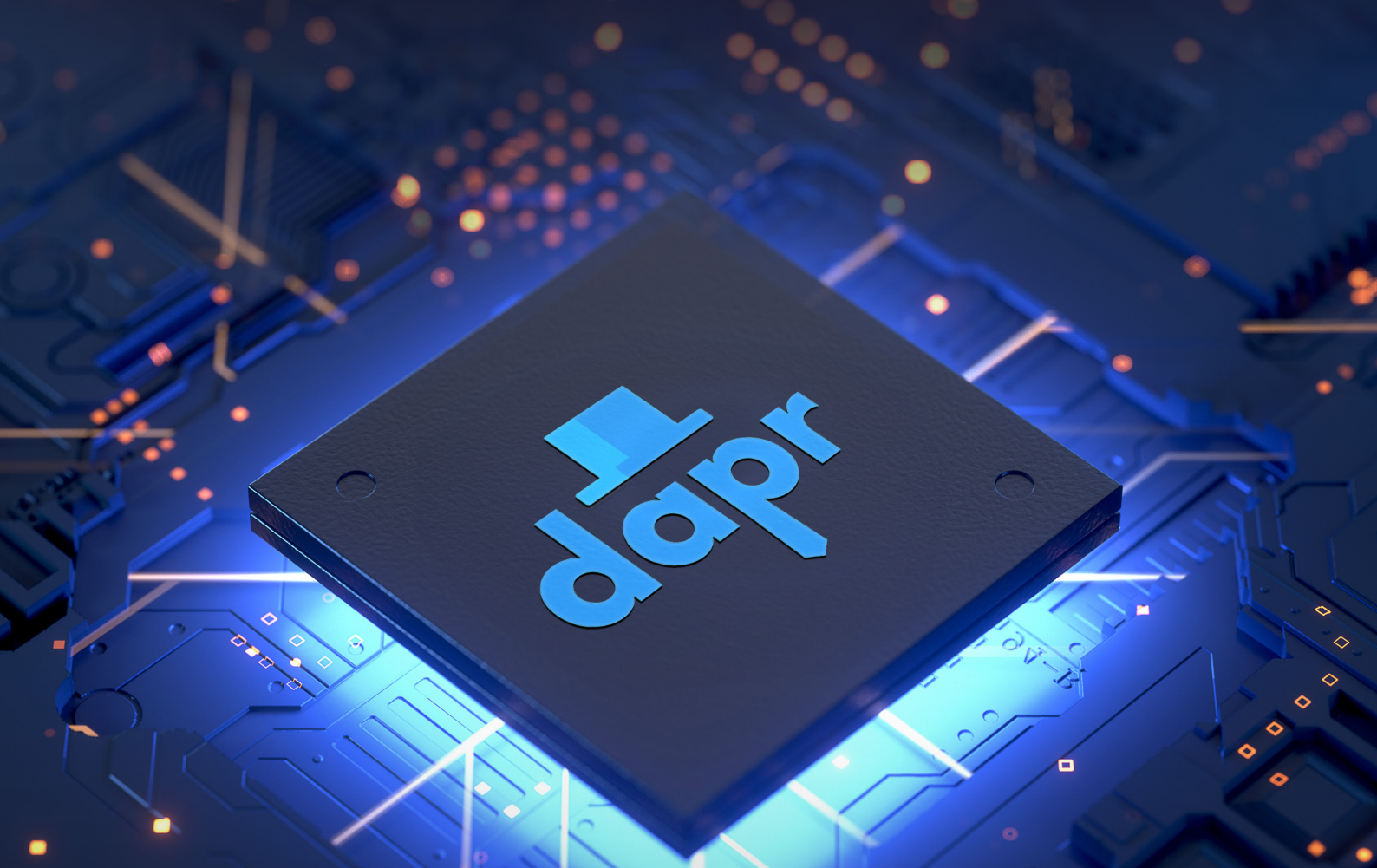
T.C. SANAYİ VE TEKNOLOJİ BAKANLIĞI

#MİLLİ TEKNOLOJİ HAMLESİ

TÜBİTAK

DAPR İLE ENTEGRE VE ÖLÇEKLENEBİLİR UYGULAMA GELİŞTİRME

ARAŞTIRMA SERİSİ - SAYI 18



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
AOP	Yön Tabanlı Programlama (Aspect Oriented Programming)
API	Uygulama Programlama Arayüzü (Application Programming Interface)
CA	Sertifika Otoritesi (Certificate Authority)
CLI	Komut Satırı Arayüzü (Command Line Interface)
CNCF	Cloud Native Computing Foundation
CRD	Özel Kaynak Tanımı (Custom Resource Definition)
DAPR	Distributed Application Runtime
gRPC	Uzaktan Yordam Çağrısı (Google Remote Procedure Call)
HTTP	HyperText Aktarım Protokolü
IoT	Nesnelerin İnterneti (Internet of Things)
mTLS	Taşıma Katmanı Güvenliği (Mutual Transport Layer Security)
RabbitMQ	Rabbit Mesaj Kuyruğu (Rabbit Message Queue)
VS Code	Visual Studio Code

Yazarlar
İhsan Baran SÖNMEZ
Serra SABAH
Zeynep Ayca TANIŞLI

Yayın Koordinatörü
Kübra ERTÜRK

Editörler
Sabiha Deniz ACUN
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.

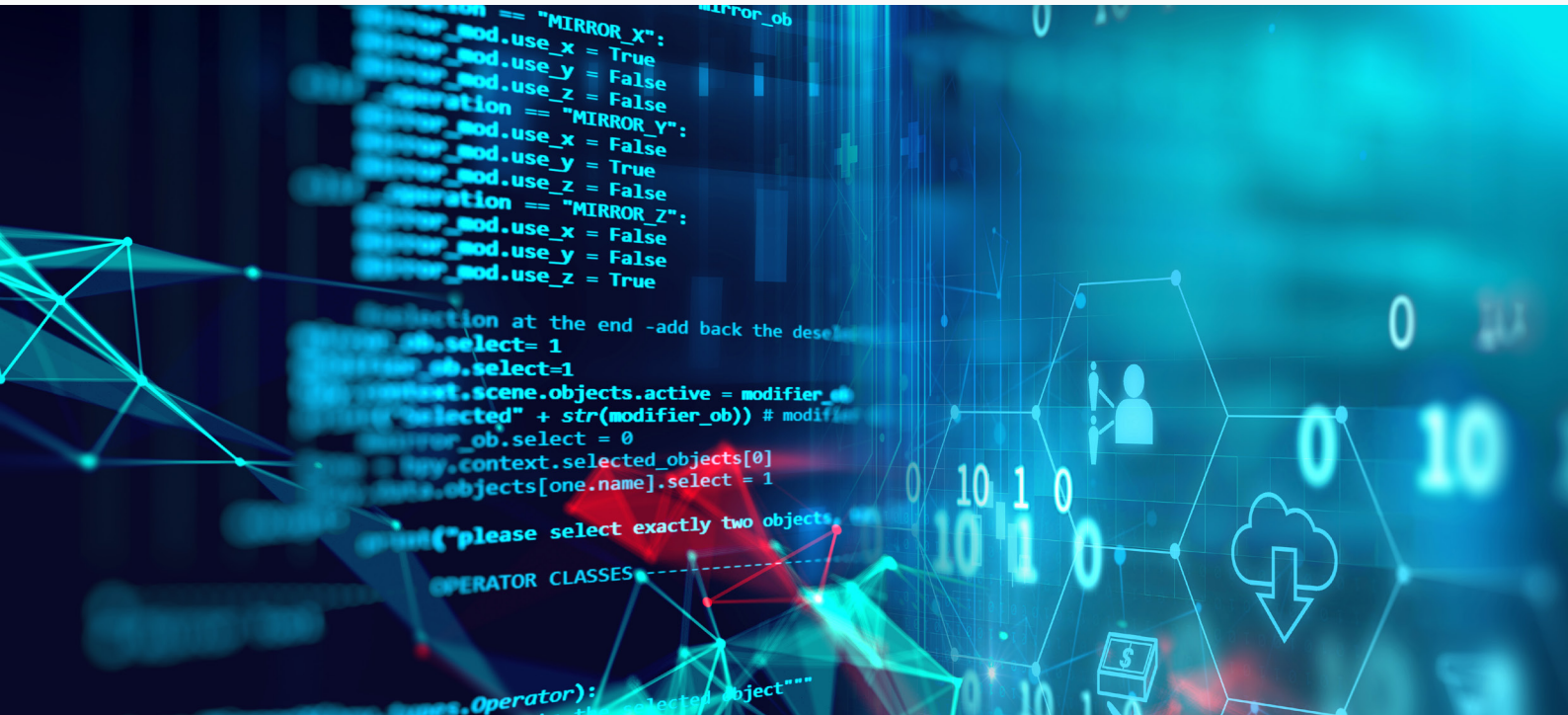
TÜBİTAK BİLGEM YTE tarafından hazırlanan Araştırma Serisi ile kurum içi içerik üretme çalışmalarının yaygınlaştırılması ve hazırlanan içeriklerin sektörün erişimine açılması amaçlanmaktadır. Araştırma Serisi'nde yayınlanan çalışmalar TÜBİTAK BİLGEM YTE çalışanlarının projelerde elde ettiği bilgi birikimini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

Giriş

Modern yazılım geliştirme dünyasında, dağıtık sistemler ve mikroservis mimarileri giderek daha fazla önem kazanmaktadır. Özellikle bulut tabanlı uygulamaların yaygınlaşmasıyla birlikte, bu tür sistemlerin karmaşıklığı ve yönetimi büyük bir meydan okuma haline gelmiştir. Dağıtık sistemler, yüksek ölçeklenebilirlik ve güvenilirlik sağlarken, aynı zamanda sistem tasarımı ve yönetimi konularında çeşitli zorluklar barındırır. Bu zorlukların üstesinden gelmek için geliştirilen Dapr (Distributed Application Runtime), açık kaynaklı bir platform olarak birçok probleme çözüm sunmayı hedeflemektedir. Aynı zamanda, mikroservis mimarisine sahip uygulamaların geliştirilmesini ve yönetilmesini kolaylaştırmak için tasarlanmış bir araçtır.

Dapr, farklı programlama dilleri ve bulut sağlayıcıları arasında taşınabilirlik sağlayarak, geliştiricilerin iş yükünü hafifletmekte ve uygulama geliştirme süreçlerini hızlandırmaktadır. Mikroservisler arasındaki iletişimin kolaylaştırılması, durum yönetiminin (state management) optimize edilmesi ve güvenli veri alışverişinin sağlanması için çeşitli bütünleşmiş çözümler sunulmaktadır. Bu çözümler; servis keşfi, mesajlaşma, izleme ve dağıtık izleme gibi kritik özellikleri içermektedir. Ayrıca, Dapr'ın olay güdümlü programlama ve aktör tabanlı modelleme gibi gelişmiş özellikleri, geliştiricilerin karmaşık iş senaryolarını daha basit ve etkili bir şekilde modellemelerini mümkün kılmaktadır. Modüler yapısı, esnek bir şekilde özelleştirme imkânı sunmakta ve teknolojik gelişmelere hızlı uyum sağlamayı kolaylaştırmaktadır.

Bu çalışmanın amacı, Dapr'ın mikroservis tabanlı uygulamalarda nasıl kullanılabileceği, sağladığı avantajlar ve neden olduğu dezavantajlar ile kullanım senaryolarını açıklamaktır. Dezavantajlar arasında platformun ek konfigürasyon gereksinimleri, geliştirici ekibin yeni araçları öğrenme sürecinin uzunluğu ve bazı senaryolarda potansiyel performans düşüşleri yer almaktadır. Ayrıca, Dapr'ın temel bileşenleri ve işleyişi hakkında detaylı bilgi verilerek, geliştiricilerin bu aracı daha etkin kullanmaları sağlanacaktır. Dapr'ın özellikleri, avantajları ve dezavantajları incelenerek, modern dağıtık uygulama geliştirme süreçlerinde nasıl bir çözüm sunduğu açıklanacaktır. Ayrıca, Dapr'ın akademik ve pratik uygulama alanları, kullanım senaryoları ve entegrasyon araçları detaylandırılacaktır.



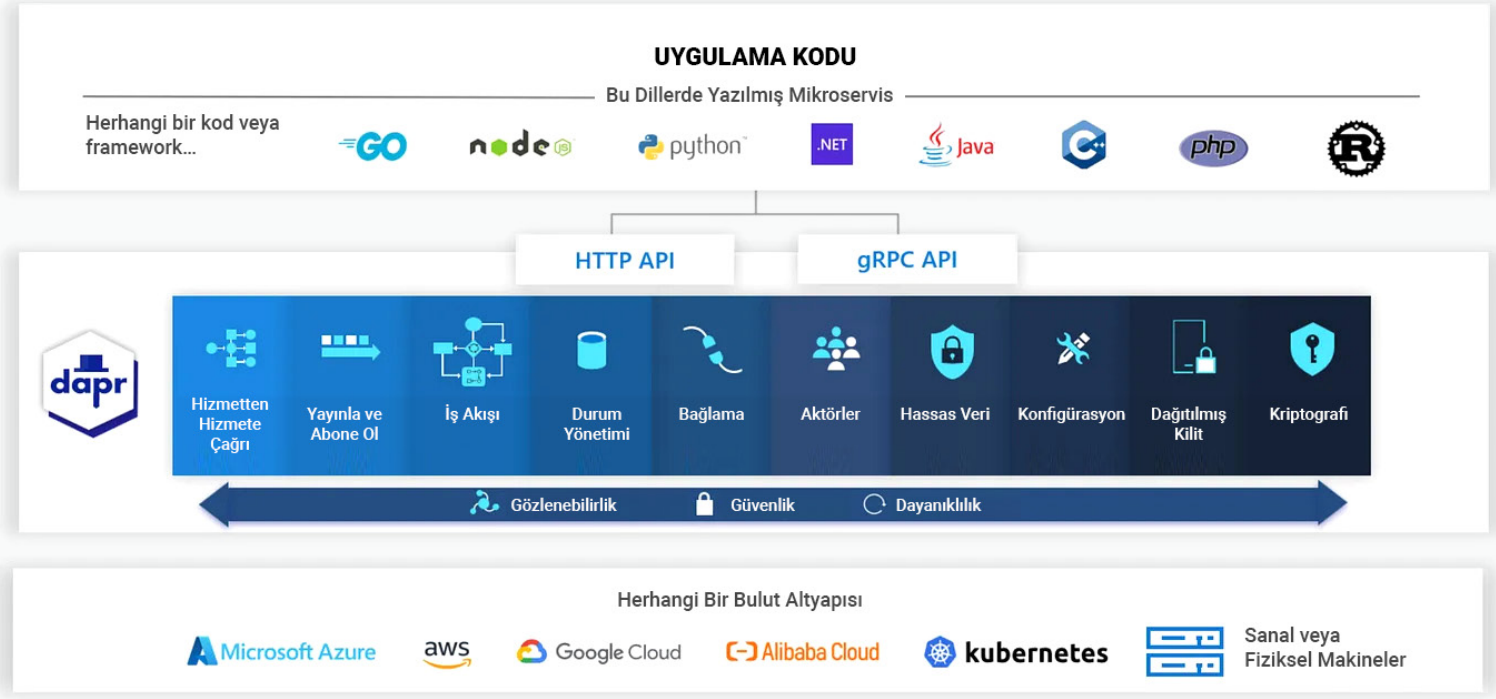
Dapr Nedir?

Günümüzde yeni veya mevcut uygulamaları geliştirmek için mikroservis mimarisi yaygın olarak kullanılmaktadır. Ancak mikroservis mimarisi ile çalışırken karşılaşılan güvenli iletişim ihtiyacı, hizmetlerin birbirini bulması ve iletişim kurması için hizmet keşfi, uygulamaların durum bilgisini yönetme gerekliliği, mikroservislerin bağımsız olarak çalışabilmesi için hizmetlerin ayrılması ve tüm bu süreçlerin izlenebilir ve gözlemlenebilir olması gibi zorluklarla karşılaşılabilir.

Dapr, bulutta ve uçta çalışan uygulamaları geliştirmeyi kolaylaştıran, taşınabilir ve olay odaklı bir çalışma zamanıdır. Mikroservis mimarisine özgü zorlukların üstesinden gelmek amacıyla ortaya çıkan Dapr, geliştiricilere dayanıklı, esnek ve taşınabilir mikroservisler oluşturma imkânı tanır. HTTP ve gRPC çağrıları aracılığıyla erişilebilen bir API sunarak, hizmetler arası iletişim, durum yönetimi, güvenlik ve izleme gibi kritik alanlarda çözümler sağlar.

Dapr, 16 Ekim 2019'da Microsoft tarafından başlatılmış ve duyurulmuş olup, şu anda Cloud Native Computing Foundation (CNCF) tarafından geliştirilen ve GitHub'da açık kaynaklı olarak sunulan bir projedir. Bu sayede, Dapr'ın sunduğu çözümler geniş bir geliştirici kitlesi tarafından kullanılabilir ve sürekli olarak geliştirilmektedir.

Mikroservislerin yönetimi ve birbirleriyle iletişimi, geliştiriciler için ciddi zorluklar ortaya çıkarmaktadır. Hizmetler arası iletişim, durum yönetimi, güvenlik, izleme ve hata toleransı gibi konular, mikroservis tabanlı mimarilerde sıkça karşılaşılan zorluklardır. Bu zorluklar, uygulama geliştirme sürecini yavaşlatabilir ve hata yapma olasılığını artırabilir.



Şekil 1. Dapr Mimarisi

Şekil 1, Dapr'ın temel mimarisini ve nasıl çalıştığını görsel olarak özetlemektedir. Dapr, Kubernetes üzerinde, belirlenmiş makinelerde, IoT (Nesnelerin İnterneti) cihazlarında veya konteyner olarak çalışabilen esnek bir yapıya sahiptir. Uygulamalar açısından Dapr, HyperText Aktarım Protokolü (HTTP), Uzaktan Yordam Çağrısı (gRPC) çağrıları veya SDK'lar aracılığıyla doğrudan erişilebilen bir API sunar. Dapr, uygulama koduyla HTTP veya gRPC API'leri aracılığıyla iletişim kurar ve kullanıcıların çerçeveleri veya kitaplıkları içe aktarmasına gerek kalmadan mevcut ve gelecekteki tüm programlama dillerini destekler. Herhangi bir dil veya çerçeveyle çalışabilme özelliği sayesinde **.NET, Java, Go, Python, PHP, JavaScript, C++** ve **Rust** gibi birçok programlama dili için kullanılabilir. Böylece geliştiriciler dillerini ve tercih ettikleri çerçeveleri kullanarak güvenli, etkili ve yönetilebilir dağıtılmış uygulamalar geliştirebilirler.

Dapr, mikroservislerin yönetimini kolaylaştırmak için bir dizi yerleşik özellik sunmaktadır. Bu özellikler arasında servis keşfi, durum yönetimi, mesajlaşma ve izleme bulunmaktadır. Dapr bu özellikleri sağlarken, geliştiricilerin dağıtık sistemlerin karmaşıklıklarıyla uğraşmadan iş mantığına odaklanmalarını sağlar. Dapr, mikroservis tabanlı uygulamaların geliştirilmesini ve yönetilmesini önemli ölçüde kolaylaştırmaktadır. Özellikle geliştiricilerin dağıtık sistemlerin karmaşıklıklarıyla uğraşmadan iş mantığına odaklanmalarını sağlamaktadır. Dapr'ın sağladığı modüler ve genişletilebilir yapı, geliştiricilere büyük esneklik sunar. Dapr, mevcut mikroservis yönetim araçlarına kıyasla daha modüler ve genişletilebilir bir yapı sunmaktadır. Kubernetes ve Istio gibi araçlar, mikroservislerin yönetiminde önemli rol oynarken, Dapr daha hafif ve taşınabilir bir alternatif sunar. Ayrıca Dapr'ın platformdan bağımsız olması önemli bir avantajdır.

Dapr'ın Ortaya Çıkışı

Dapr, Microsoft tarafından 2019 yılında açık kaynak olarak tanıtıldı. Microsoft, Azure bulut platformu üzerindeki deneyimlerinden ve kullanıcı geri bildirimlerinden yola çıkarak, mikroservislerin geliştirilmesini kolaylaştıracak bir çözüm geliştirdi. Açık kaynak topluluğunun katkılarıyla Dapr, hızlı bir şekilde genişleyip olgunlaştı.

Dapr'ın Temel Amaçları

Dapr, dağıtık uygulama geliştirme sürecinde geliştiricilerin karşılaştığı zorlukları azaltmayı amaçlayan bir platformdur. Bu amaç doğrultusunda Dapr, üç temel hedefe odaklanır:

1. Dağıtık Uygulama Geliştirmeyi Kolaylaştırma

Dapr, geliştiricilerin dağıtık sistemlerin karmaşıklıklarıyla uğraşmak yerine iş mantığına odaklanmalarına olanak tanır. Bu, daha hızlı ve verimli bir geliştirme süreci sağlar. Dapr'ın modüler yapısı, farklı bileşenlerin kolayca eklenip çıkarılabilmesine olanak tanır. Bu esneklik, Dapr'ın geniş bir kullanım alanı bulmasını sağlamıştır. Geliştiriciler, mevcut bileşenleri özelleştirebilir veya yeni bileşenler ekleyebilir, bu da Dapr'ın çeşitli kullanım senaryolarına uyarlanabilir olmasını sağlar.

2. Çapraz Platform ve Bulut Uyumu

Dapr, farklı bulut platformları ve ortamlarla uyumlu çalışır. Bu, uygulamaların belirli bir bulut sağlayıcısına bağımlı olmadan taşınabilirliğini artırır. Ek olarak, geliştiricilerin uygulamalarını çeşitli bulut ortamlarında ve yerel geliştirme ortamlarında çalıştırabilmelerini sağlar.

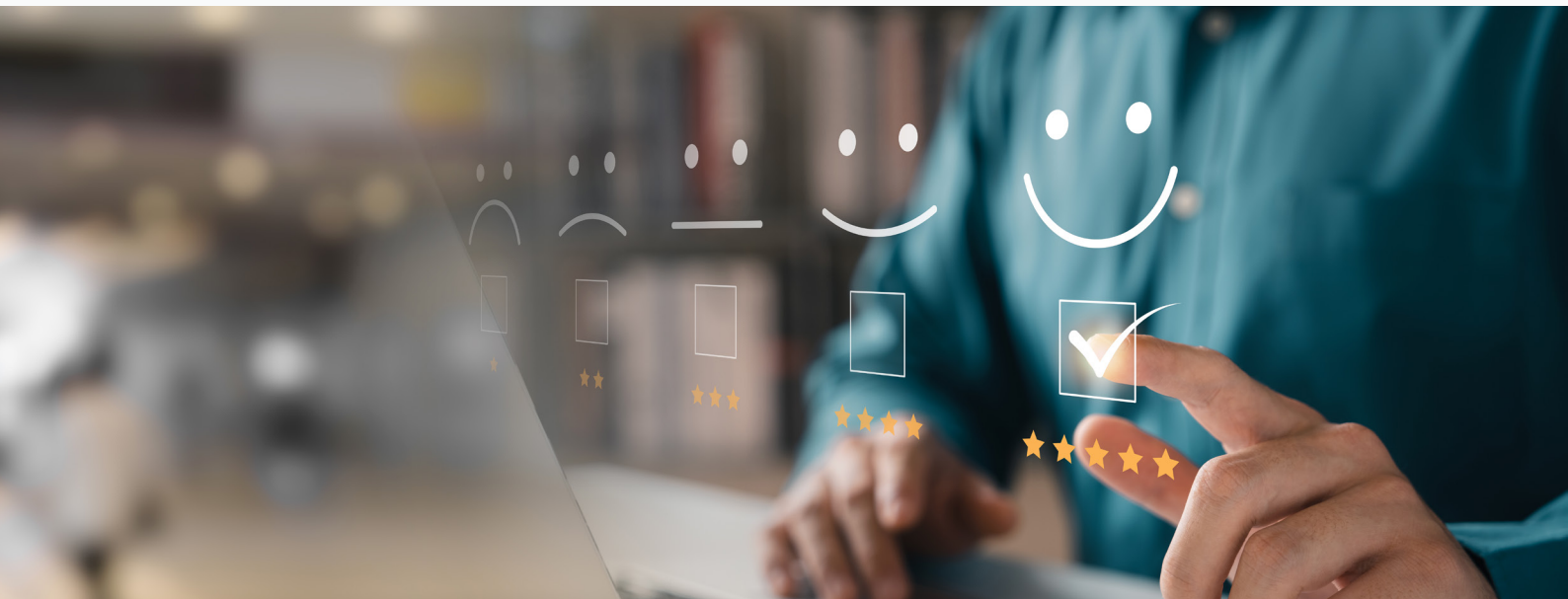
3. Dayanıklılık (Resiliency) ve Güvenilirlik (Reliability)

Dapr, dağıtık sistemlerde ortaya çıkan hata toleransı ve dayanıklılık gereksinimlerini ele alır ve bu konudaki zorlukları hafifletir. Dapr, bu tür zorlukları standart kütüphaneler ve API'ler aracılığıyla çözmeyi hedefler, böylece geliştiricilerin bu konularla ilgili endişelerini azaltır.

Dapr'ın Avantajları ve Dezavantajları

Avantajları

- **Kolay Entegrasyon ve Standartlaştırma:** Mikroservislerin arasındaki durum yönetimi, mesajlaşma ve servis keşfi gibi yaygın fonksiyonları, standart API'ler aracılığıyla sağlar.
- **Platform ve Dil Bağımsızlığı:** Herhangi bir programlama dili ve platformla çalışabilir.
- **Yan Bileşen Modeli:** Her mikroservisin yanında çalışan bir yan bileşen olarak yapılandırılır.
- **Gözlemlenebilirlik ve İzlenebilirlik:** Yerleşik telemetri, izleme ve hata ayıklama özellikleri sunar.
- **Çapraz Bulut Uyumluluğu:** Çeşitli bulut sağlayıcılar ve yerel ortamlarla uyumlu çalışabilir.
- **Hızlı Geliştirme ve Prototipleme:** Geliştirme sürecini hızlandırmak için bir dizi hazır bileşen ve API sunar.
- **Açık Kaynak ve Topluluk Desteği:** Geniş bir topluluk tarafından desteklenir ve sürekli olarak geliştirilir.



Dezavantajları

- **Ek Öğrenme Eğrisi:** Mikroservis geliştirme konusunda yeni olan geliştiriciler için öğrenme gerektirir.
- **Dağıtık Sistemlerin Karmaşıklığı:** Dağıtık sistemlerin karmaşıklığını azaltmayı hedeflese de tamamen ortadan kaldıramaz.
- **Sisteme Fazla Yüklenmesi:** Yan bileşen modeli, bazı durumlarda performans ve kaynak kullanımı üzerinde ek bir yük yaratabilir.
- **Destek ve Dokümantasyon Eksiklikleri:** Bazı bileşenleri ve özellikleri için sınırlı belgelendirme ve topluluk desteği bulunabilir.
- **Bağımlılıklar:** Mevcut sistemlere ek bir bağımlılık ekler ve bu bazı projelerde karmaşıklığı artırabilir.

Dapr ile Uygulama Geliştirme Araçları ve Entegrasyonlar

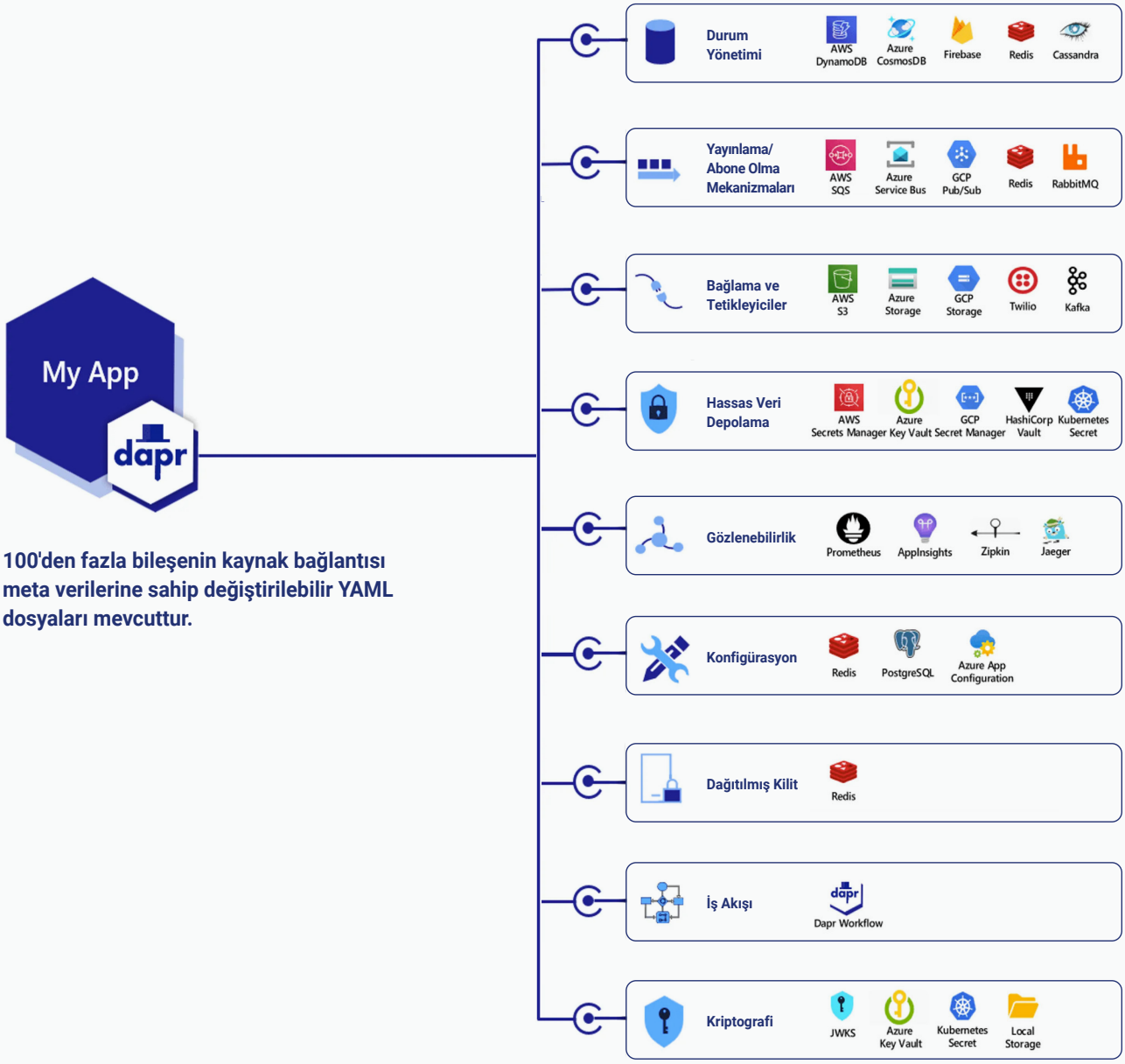
Dapr, mikroservis tabanlı uygulamaların geliştirilmesini ve yönetilmesini kolaylaştırırken, aynı zamanda çeşitli geliştirme araçları ve entegrasyonlarla uyumlu çalışabilen esnek bir platform sunar. Bu bölümde, Dapr ile kullanılabilecek önemli uygulama geliştirme araçları ve entegrasyon yöntemleri ele alınacaktır.

Temel Araçlar

1. **Dapr CLI:** Dapr'ı yerel olarak kurmak, yapılandırmak ve yönetmek için kullanılır.
2. **Kubernetes (K8s):** Dapr, Kubernetes üzerinde çalıştırılabilir ve yönetilebilir.
3. **Visual Studio Code (VS Code):** VS Code için Dapr eklentisi, Dapr ile çalışan uygulamaları geliştirmeyi kolaylaştırır.
4. **Docker:** Dapr uygulamalarının yerel geliştirme ortamında konteynerler içinde çalıştırılmasını sağlar.
5. **Dapr Gösterge Paneli (Dashboard):** Dapr bileşenlerini ve uygulamalarını görselleştirmek için kullanılır.

Uygulama Entegrasyonları

1. **Durum Yönetimi (State Management) ve Mesajlaşma (Publish/Subscribe):** Dapr, durum yönetimi ve mesajlaşma için HTTP ve gRPC API'leri sağlar.
2. **Bağlantılar ve Yayınla/Abone Ol (Bindings and Pub/Sub):** Dapr, çeşitli veri kaynakları ve hizmetlerle etkileşime geçmek için bağlantılar ve yayınla/abone ol modelini destekler.
3. **Gözlemlenebilirlik (Observability):** Dapr, izleme ve metrik toplama sistemleri ile entegre olabilir.

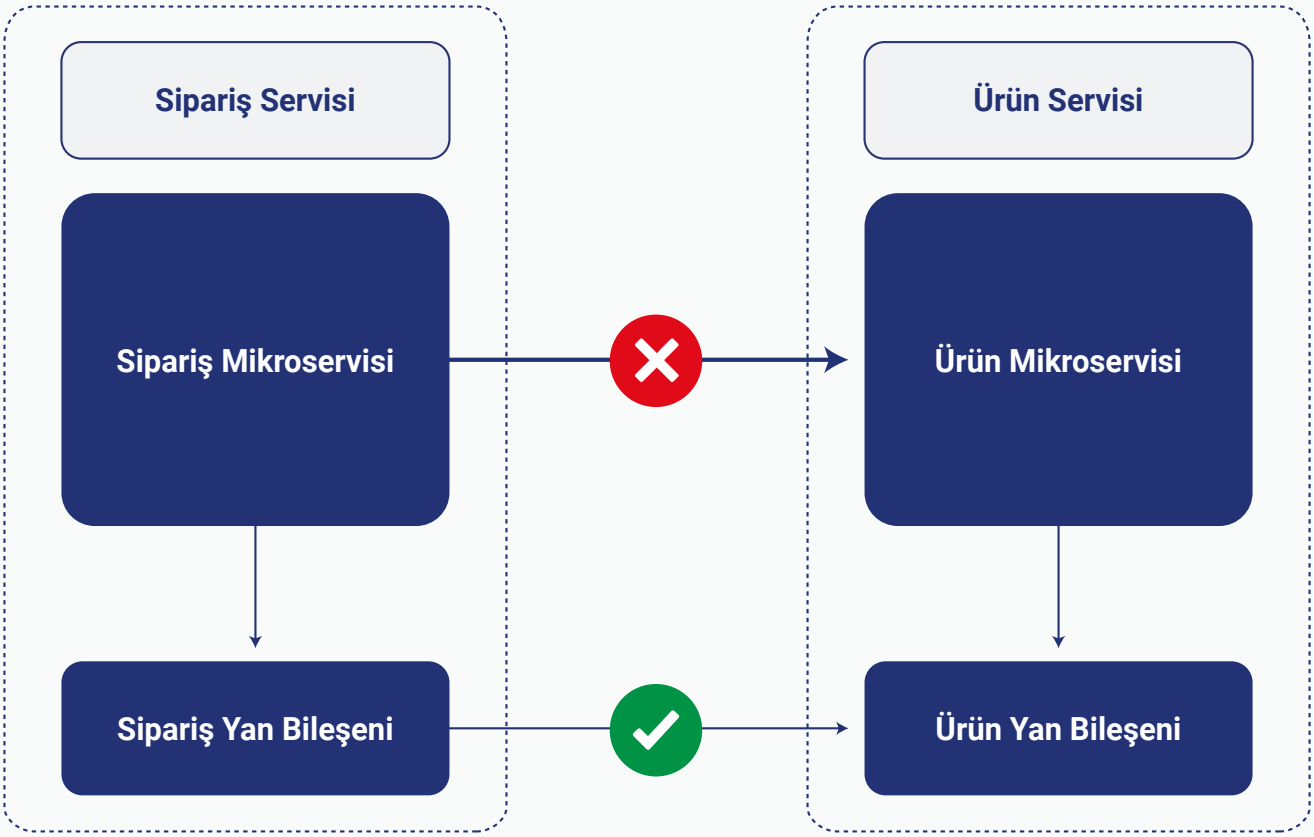


Şekil 2. Dapr Bileşenleri ve Dağıtılmış Uygulamaları

Şekil 2, Dapr'ın bileşenlerini ve bunların dağıtılmış uygulamalardaki işlevlerini genel olarak özetlemektedir. Bu bileşenlerin bir araya gelmesi, Dapr'ın dağıtılmış sistemlerdeki karmaşıklığı azaltarak uygulama geliştirme sürecini daha verimli hale getirmeyi hedefler.

Dapr Yan Bileşeni (Sidecar)

Dapr, uygulamaların ve mikroservislerin yanında çalışan ayrı bir süreç olarak konumlandırılan bir "yan bileşen" yaklaşımını ele alır. Her mikroservis için bir Dapr yan bileşeni bulunur. Dapr uygulamaları, Dapr'ın yapı taşlarını kullanmak istediğinde bu yapı bileşen ile basit ağ istekleri aracılığıyla iletişim kurar. Ayrıca, Dapr'ın gRPC kullanarak yapı bileşenleri arasındaki iletişimi optimize eder.



Şekil 3. Dapr Yan Bileşeni

Şekil 3, bir sipariş ve ürün mikroservisi arasındaki yapı ve ilişkileri görsel olarak temsil etmektedir. Şekilde, her mikroservisin yanında bulunan yan bileşenler, bu servislerin işlevlerini destekleyerek bütünleşmiş bir dağıtım yapısının nasıl çalıştığını ve mikroservislerin nasıl etkileşimde bulunduğunu göstermektedir.

Bu yaklaşımın bazı faydaları şunlardır:

- **Güvenlik:** Dapr, mikroservisler arasındaki trafiği otomatik olarak şifreleyebilecek yan bileşen üzerinde mTLS gibi güvenlik önlemlerini uygular.
- **Erişim Kontrolü:** Dapr, erişim kontrol politikalarını yan bileşen üzerinde uygulayarak güvenlik önlemlerini artırabilir.
- **İzleme ve Ölçümleme:** Dapr, uygulamaların performansını ve durumunu izleyebilmek için yan bileşen içinde izleme ve ölçümleme işlevselliği sağlar.

Dapr API

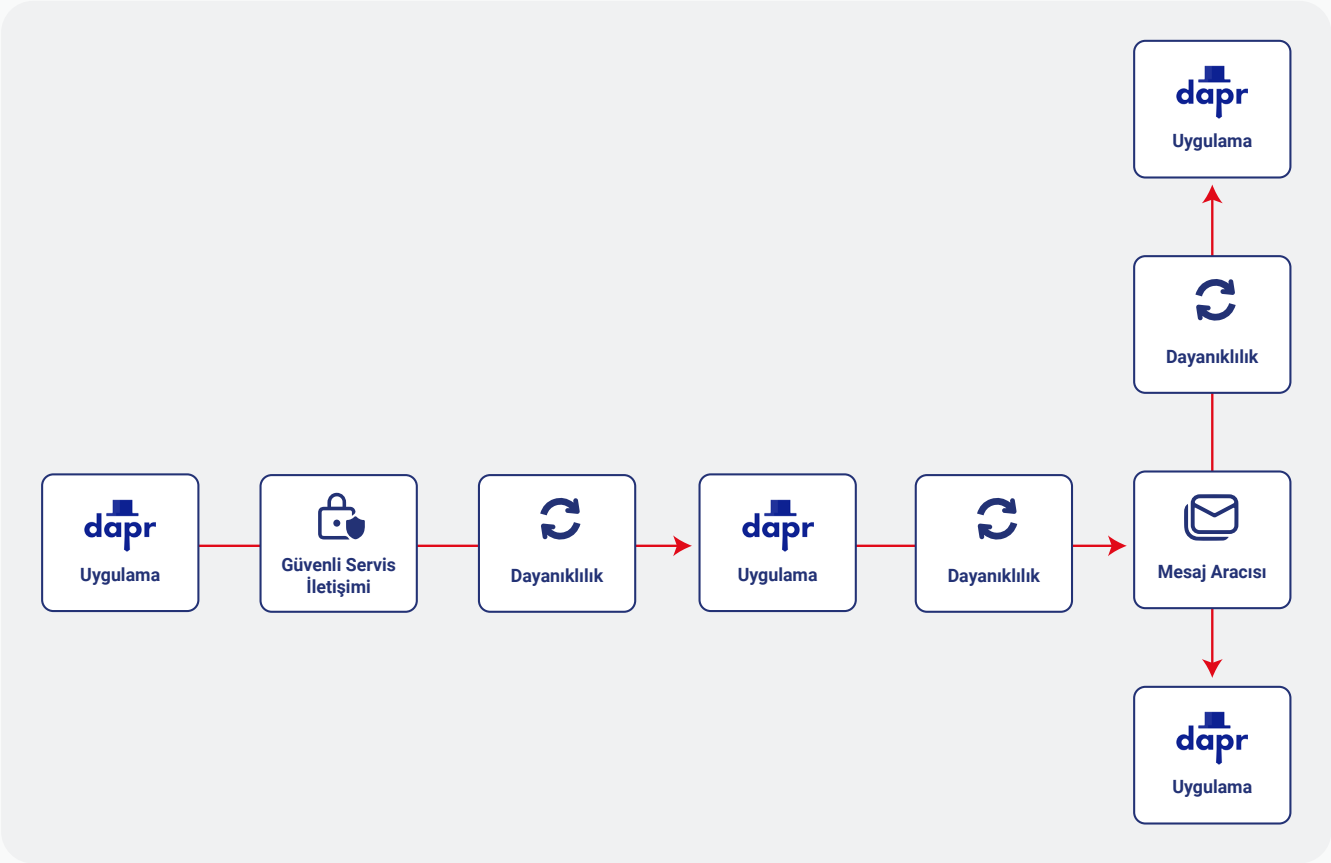
Dapr, programlama dilinden bağımsız HTTP ve gRPC API'ler sağlayan, açık kaynaklı bir dağıtılmış uygulama çalışma zamanıdır. Dapr, dağıtılmış mikroservis uygulamaları oluştururken, karşılaşılan yaygın sorunların karmaşıklığını ortadan kaldırmak için yapı taşları şeklinde çözümler sunar. Her yapı taşı, HTTP uç noktası aracılığıyla herhangi bir uygulamadan çağrılabilen API sağlar.

Dapr, sunduğu API'ler sayesinde güvenlik, dayanıklılık ve gözlemlenebilirlik açısından sektördeki en iyi uygulamalardan yararlanır. Bu sayede, geliştiriciler kodlarına odaklanabilirler. Geliştiricilerin her projede tekrar eden altyapısal kodlarını yazmadan doğrudan iş kurallarını gerçekleştirdiği üretime hazır uygulamalara ulaşmalarını sağlar.

Dapr'ın Servis Keşfi API'si, mikroservislerin birbirlerini bulmalarını ve çağırmalarını kolaylaştırırken, Durum Yönetimi API'si, uygulama durumunun merkezi bir şekilde yönetilmesini sağlar. Yayınla/Abone Ol (Pub/Sub) API'si olay tabanlı mimarilerin kolaylıkla oluşturulmasına olanak tanır ve Aktör Modeli API'si karmaşık iş mantıklarının aktör tabanlı modellerle daha basit bir şekilde gerçekleştirilmesini sağlar.

Dapr ile Güvenli ve Dayanıklı Uygulamalar

Dapr, geliştiricilere güvenli ve dayanıklı mikroservis tabanlı uygulamalar oluşturmak için önerilen en iyi araçları ve hizmetleri sunar. Bu araçlar ve hizmetler, dağıtık uygulamalarda yaygın olan zorlukları çözmeye yardımcı olur. API'ler, hizmetler ve bileşenler üzerinde uygulama merkezli güvenlik politikaları belirlenerek erişim sınırlandırılabilir. İletişim mTLS ile otomatik olarak şifrelenir ve donanım arızaları gibi olaylara karşı esneklik politikaları sağlar. Bu sayede hızlı bir geliştirme yapmanın yanında güvenliği de elden bırakmayıp, modern güvenlik standartlarını destekleyen ve içeren uygulamalar geliştirmeye yardımcı olur.

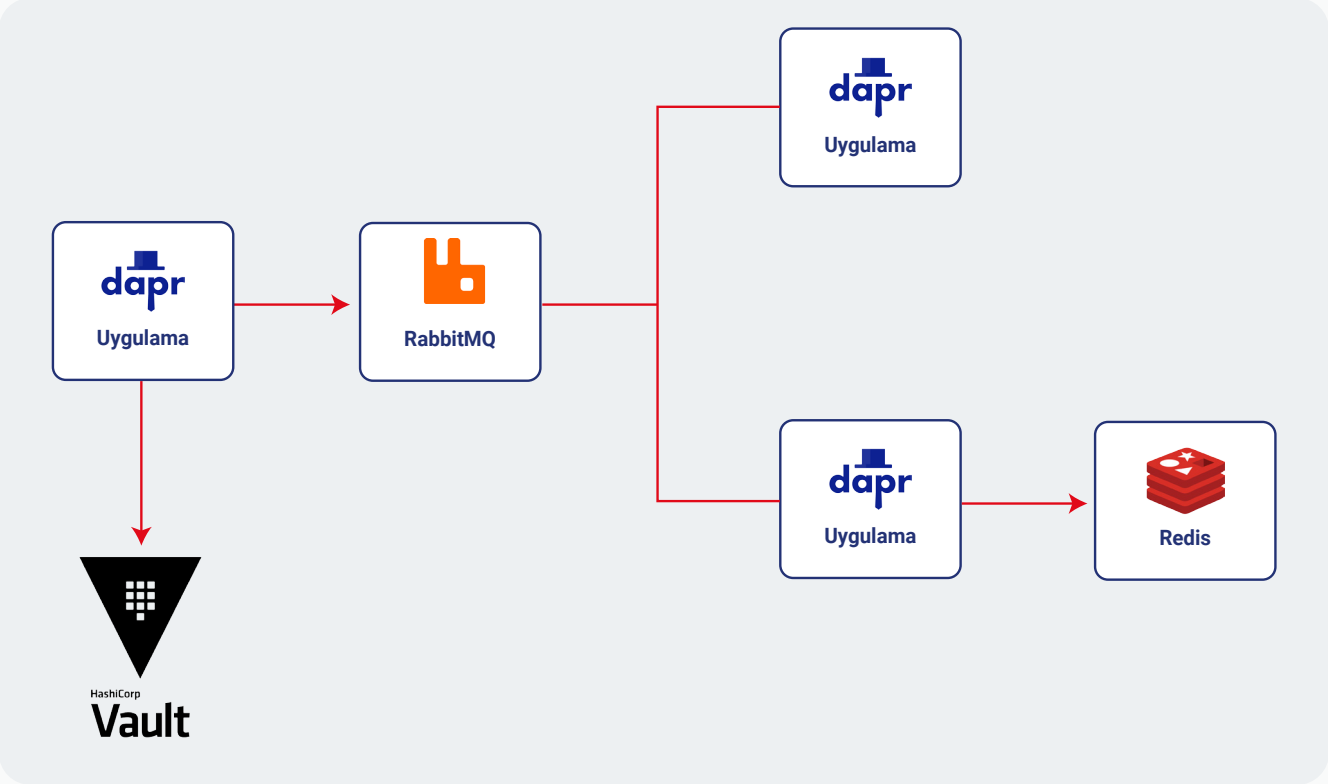


Şekil 4. Güvenli İletişim

Şekil 4'de, birbirinden bağımsız olarak çalışan uygulamalar yer almakta ve her biri Dapr ile entegre edilmiştir. İki uygulama arasındaki güvenli iletişimi sağlamak için Dapr'ın sunduğu şifreleme yöntemleri kullanılır. Ayrıca, Dayanıklılık "Resiliency" okları ile Dapr uygulamaların dayanıklılığını artırır. Bu, Dapr'ın olası hata durumlarında uygulamaların düzgün çalışmaya devam etmesini sağlar. Ayrıca, bir mesaj aracısı (message broker) yer almaktadır. Dapr bu aracının güvenli ve dayanıklı bir şekilde kullanılmasını sağlar. Uygulamalar, Dapr üzerinden bu mesaj aracısı ile iletişime geçer ve böylece sistem genelinde mesajlaşma süreci güvenli ve kesintisiz bir şekilde gerçekleştirilir.

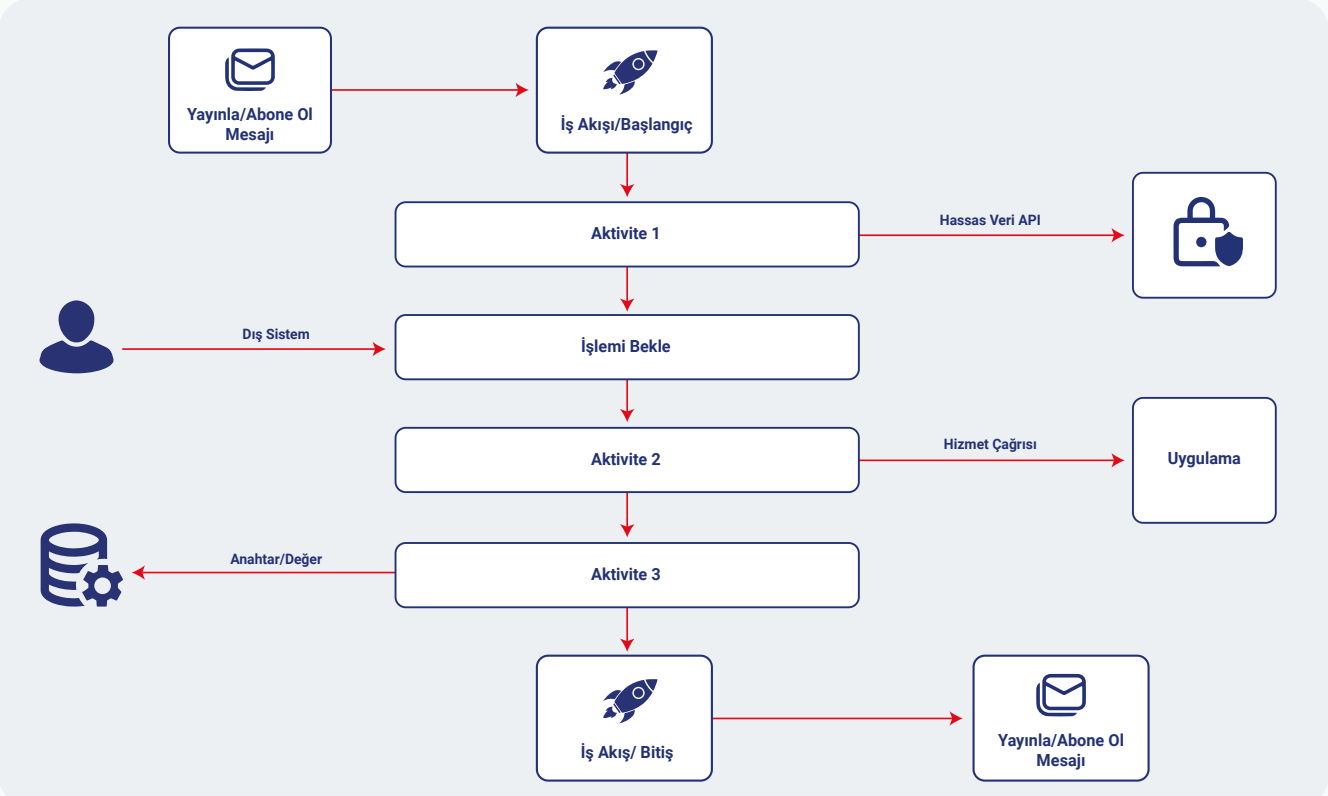
Dapr Bileşen Modeli ile Altyapılar Arasında Entegrasyon

Dapr'ın bileşen modeli, farklı altyapılarla entegrasyonu kolaylaştırır. Şekil 5'te, yayınla/abone ol işlevi RabbitMQ'dan Redis'e geçiş yapılarak değiştirilmesi gösterilmiştir. Dapr uygulamaları Kubernetes veya diğer platformlarda kendi kendini barındırabilme özelliğiyle geliştirilen uygulamaların farklı ortamlarda farklı bileşenlerle çalışmasını sağlar. Dapr, yaml formatında tanımlanan konfigürasyon dosyaları sayesinde ortama göre ilgili altyapısal bileşen ile iletişime geçer. Uygulama kodunda herhangi bir değişiklik yapmaya gerek duymadan herhangi bir aracıya geçiş yapabilmesiyle geliştiricilere avantaj sağlar.



Şekil 5. On-Prem Sunucularda Dapr Örneği

Dapr ile İş Süreçlerini Otomatikleştirme

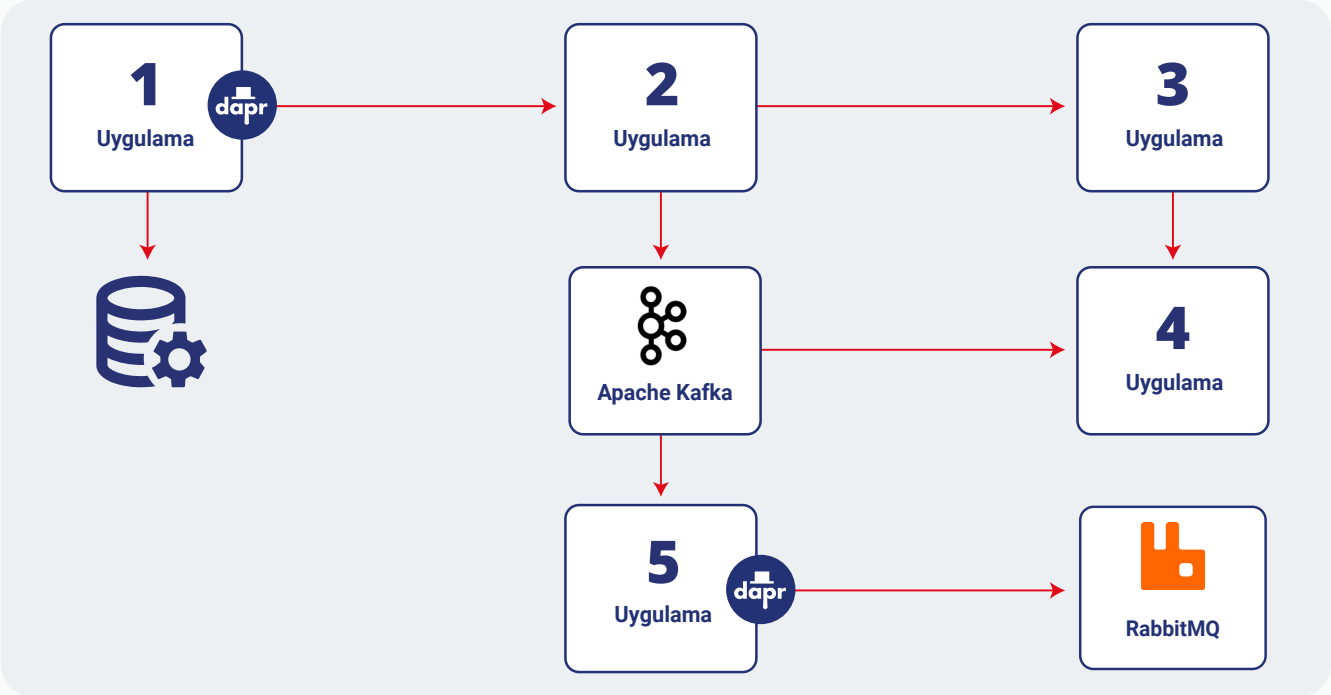


Şekil 6. Dapr İş Süreçlerini Otomatikleştirme

Dapr, dağıtılmış uygulamaların geliştirilmesini hızlandırır ve iş süreçlerini otomatikleştirir. Durum bilgisi olan karmaşık iş akışları, iş akışı modelleri kullanılarak kodlanabilir ve bu iş akışları, seçilen veri tabanında saklanmak üzere Dapr API'leriyle entegre edilebilir. Ayrıca, bu iş akışlarının çalışma durumlarının görüntülenmesi ve mevcut akışların takibi konusunda da yardımcı olur.

Dapr'ın Mevcut Mimarilere Entegrasyonu

Dapr, mevcut mimariye aşamalı olarak dahil edilebilir. Dapr sistemler arasında temelde HTTP veya gRPC yöntemlerini kullandığı için tek seferde tüm sistemin Dapr'a geçirilmesine gerek yoktur. Sistem parçalanıp zaman içinde adım adım geçirilebilir. Bu sayede mevcut uygulamaların da Dapr'a geçişi oldukça kolay olmaktadır. Herhangi bir yerin bozulmadığından emin olunarak, mevcut sisteme zarar verilmeden aşama aşama yeni yapıya geçiş gerçekleştirilebilir.



Şekil 7. Parçalı Şekilde Dapr'a Geçiş

Şekil 7'deki örnekte, hem Dapr kullanan hem de Dapr kullanmayan uygulamaların birlikte çalışabildiği gözlemlenmektedir.

Dapr, mevcut mikroservis mimarilerine entegrasyon sağlarken, verimli bir veri akışı ve iletişim yönetimi sunar. Örneğin, beş uygulamadan oluşan bir sistemde, Kafka ve RabbitMQ'nun nasıl entegre edileceğini ele alınabilir.

Uygulama 1 verileri toplar ve işlemi başlatır. Dapr'ın **publish** API'sini kullanarak, bu verileri Kafka'ya yayınlar.

Uygulama 2, Kafka'dan gelen verileri almak için Dapr'ın **subscribe** API'sini kullanarak bu verileri alır ve işler. Bu işlenmiş veri, **Uygulama 3** tarafından daha ileri bir işlem için kullanılır.

Uygulama 3, veriyi hazırlayarak bir sonraki uygulama olan **Uygulama 4**'e iletir.

Uygulama 4, veriyi finalleştirir ve RabbitMQ'ya gönderir. Bu noktada, Dapr'ın **publish** API'si, veriyi RabbitMQ'ya aktarmak için kullanılır.

Uygulama 5, RabbitMQ'dan gelen veriyi alır ve son işleme tabi tutar. Bu süreçte Dapr'ın **subscribe** API'si, App 5'in RabbitMQ'dan veri almasını sağlar. Bu yapı, Dapr'ın **publish** ve **subscribe** API'lerini kullanarak veri akışını verimli bir şekilde yönetir, Kafka ve RabbitMQ'nun rollerini net bir şekilde tanımlar ve mikroservislerin etkili bir şekilde entegrasyonunu sağlar.

Dapr Yapı Taşları (Building Blocks)

Dapr, mikroservis oluşturulmasına yönelik ortak en iyi uygulamaları sağlayan yedi modüler yapı taşı sunar. Bu yapı taşları, mikroservislerin çeşitli işlevlerini etkin bir şekilde yönetmek ve geliştirmek için tasarlanmıştır. Durum Yönetimi, verilerin kalıcı olarak saklanmasını ve erişilmesini sağlar. Servis Çağırma, mikroservisler arasında güvenli ve verimli iletişim kurar. Yayınlama ve abone olma, asenkron mesajlaşma ve olay tabanlı iletişimi destekler. Gözlemlenebilirlik, uygulama performansını izler ve hata ayıklamayı kolaylaştırır. Bağlamalar, dış sistemlerle (external system) etkileşim kurmayı sağlar. Yapılandırma, merkezi bir yapıdan uygulama ayarlarını yönetir. Son olarak, Kimlik Doğrulama ve Yetkilendirme, gizli bilgilerin güvenli bir şekilde saklanmasını ve erişilmesini temin eder. Bu yapı taşları, mikroservislerin etkili bir şekilde entegrasyonunu ve yönetimini sağlar, bu da sistemin genel verimliliğini artırır. Tamamen bağımsız olan yapı taşları, geliştiricilere uygulama gereksinimlerine bağlı olarak istedikleri veya sadece ihtiyaç duydukları yapı taşlarını seçip kullanma esnekliği sunar.

Şekil 8. Dapr Yapı Taşları

Dapr, uygulamalara yeni özellikler ve yetenekler eklemek için kullanılabilecek çeşitli API yapı taşlarından oluşur. Bu API yapı taşları, geliştiricilere mikroservislerini daha kolay ve etkili bir şekilde oluşturma, yönetme ve ölçeklendirme imkanı tanır. Şekil 8'deki tabloya bakıldığında, durum yönetimi API'si, veri tabanı işlemleri gibi karmaşık durum yönetimi işlemlerini basitleştirirken, servis çağırma API'leri, mikroservisler arasında güvenli ve güvenilir bir şekilde iletişim kurmayı sağlar. Ayrıca, yayınla ve abone ol mekanizmasını destekleyen API'ler, mikroservislerin birbirleriyle asenkron olarak iletişim kurmasına olanak tanırken, bağlamalar, harici sistemlerle etkileşimi kolaylaştırır. Gözlemlenebilirlik API'leri ise, uygulamaların performansını izlemeyi ve sorunları hızlı bir şekilde tespit etmeyi sağlar. Bu API yapı taşları, Dapr'ı mikroservisler mimarisi ile geliştirilen uygulamalar için güçlü ve esnek bir çözüm haline getirir.

1. Hizmetten Hizmete Çağrı (Service-to-Service Invocation)

Dapr, mikroservislerin birbirleriyle HTTP veya gRPC mesajları aracılığıyla iletişim kurmasını sağlayan bir hizmetten hizmete çağrı özelliği sunar. Bu sayede, Dapr hizmetlerin bulundukları ortamda sorunsuz bir şekilde her yerde çalışmasını sağlar. Hizmet çağırma özelliği, mikroservislerin altta yatan ağ karmaşıklıklarını ortadan kaldırarak iletişimlerini basitleştirir. Dapr, hizmetler arasındaki iletişimi kolaylaştırmak için ters proxy (vekil sunucu) olarak bir yardımcı araç kullanır. Örneğin, 'Servis A'nın 'Servis B'ye bir istek iletmesi gerektiğinde, Dapr bu iletişimi Hizmet Çağırma API'si aracılığıyla yönetir ve hem Servis A hem de Servis B adına isteği iletir. Bu sayede, doğrudan ilgili servisi çağırmanın karmaşıklığını ortadan kaldırır.

2. Yayınla ve Abone Ol (Publish and Subscribe)

Dapr, bir mesajlaşma modeli olan yayınla/abone ol (publish/subscribe) sistemini destekleyerek mikroservislerin olay odaklı mimaride etkin iletişim kurmasını sağlar. Bu modelde, göndericiler (yayıncılar) belirli bir konuya mesaj yayınlar ve alıcılar (aboneler) bu konulara abone olarak ilgili mesajları alır. Dapr'ın pub/sub mekanizması, hizmetler arasında asenkron iletişimi kolaylaştırır ve ölçeklenebilirliğini artırır. Bu noktada sağladığı API ile beraber, olay odaklı mimaride geliştirmenin ve düşünmenin ekstra efor istediği outbox deseni gibi olay odaklı mimariye özgü kavramları da içerir. Bu sayede geliştiricinin olay odaklı mimariye sahip uygulamayı, daha kolay bir şekilde ve daha eforla geliştirmesini sağlar.

3. İş Akışı (Workflows)

İş akışı API'si, dağıtılmış sistemlerde karmaşık ve çok adımlı süreçlerin düzenlenmesine olanak tanır. Geliştiriciler, çeşitli mikroservislerin iş birliği içinde çalışmasını sağlayarak belirli iş süreçlerini tanımlayabilir ve yönetebilir. Bu API, diğer Dapr API'leriyle entegre edilebilir.

4. Durum Yönetimi (State Management)

Dapr'ın durum yönetimi özellikleri, uygulama durumunu yönetmek için çok yönlü bir çözüm sunar. Geliştiriciler, Dapr'ın durum yönetimi API'leriyle uygulama kodlarını değiştirmeden durum bilgisi depolayabilir ve erişebilir. Dapr, Redis, Azure Cosmos DB, Azure SQL Server, Cassandra, AWS, DynamoDB, Google Cloud Spanner ve PostgreSQL gibi yapılandırma yoluyla takılabilen depoları

destekler. Bu durum depoları, uygulamanın ihtiyacına göre seçilebilir ve değiştirilebilir. Bu özellik, dağıtılmış sistemlerde veri yönetimi zorluklarını azaltarak uygulamaların genel ölçeklenebilirliğini ve dayanıklılığını artırır. Örneğin lokal ortamda PostgreSQL kullanılırken, canlı sistemde Azure Cosmos DB kullanılmasını kolayca sağlar. Sadece bir konfigürasyon dosyası değişimi sayesinde ortamdan ortama farklı bileşenlerin kullanılmasına olanak tanır.

5. Bağlamalar (Bindings)

Dapr'ın Bağlama özelliği, mikroservislerin farklı kaynaklarla entegrasyonunu basitleştirir. Dapr bağlama API'si, kullanıcıların standart kod yazma ihtiyacını ortadan kaldırarak, mesajlaşma sistemlerine, API'lere ve diğer harici kaynaklara kolayca bağlanmalarını sağlar. Bu özellik sayesinde Dapr, uygulamaların harici hizmetlerle çift yönlü bağlantı kurmasını mümkün kılar. Kaynak bağlamaları ve tetikleyiciler, veri tabanları, kuyruklar gibi çeşitli harici kaynaklarla olay almayı ve göndermeyi sağlar. Örneğin, uygulamanız RabbitMQ'ya bir mesaj gönderebilir ve bu mesajı Redis gibi bir veri tabanına yazabilir.

6. Aktörler (Actors)

Dapr'ın Aktörler özelliği, iş akışı organizasyonunu kolaylaştırmak için planlama yetenekleri sunar. Dapr sayesinde, aktör modeli farklı dillerde veya platformlarda kullanılabilir. Aktörler, uygulamanın gereksinimlerine göre dinamik olarak etkinleştirilir. Aktörler, işlenmeyen (yani kullanılmayan) durumlarda bellekten kaldırılır. Bu, aktörlerin geçici olarak bellekte tutulduğu anlamına gelir. Örneğin bir düğüm arızalandığında, Dapr otomatik olarak etkinleştirilmiş aktörleri sağlıklı düğümlere taşır.

7. Hassas Veriler (Secrets)

Dapr, uygulamalar için hassas verileri güvenli bir şekilde yönetebilmelerini sağlayan bir yapı taşı API'si sunar. Bu API, genel bulut depoları, yerel depolar ve Kubernetes gibi gizli depolarla entegre olarak çalışabilir. Dapr'ın Hassas Veri Yönetimi, API anahtarları şifreler. Bu sayede diğer hassas bilgilerin güvenli bir şekilde işlenmesini sağlar. Dapr, hassas verileri uygulamalarından güvenli bir şekilde ayırıp farklı gizli depolardan kolayca yönetebilmesiyle güvenlik standartlarını yüksek tutar.

8. Konfigürasyon (Configuration)

Dapr'ın Yapılandırma API'si, desteklenen yapılandırma depolarından uygulama yapılandırma öğelerini almanızı ve bunlara abone olmanızı sağlar. Böylece yapılandırma deposunda herhangi bir değişiklik olduğunda, abone olunan uygulamalar bu değişikliklerden otomatik olarak güncellenir. Dapr'ın Yapılandırma API'si, uygulamalarınızın yapılandırma verilerini dinamik olarak yönetmesini ve güncellemesini sağlar. Consul, etcd, Kubernetes ConfigMaps gibi çeşitli yapılandırma depolarını destekler.

9. Dağıtılmış Kilit (Distributed Lock)

Dapr'ın Dağıtılmış Kilit API'si, birden fazla uygulama örneğinin kaynağa çakışmasız erişimini sağlamak için kullanılır. Bu API, özellikle dağıtılmış sistemlerde kaynakların tutarlı ve güvenli bir şekilde yönetilmesini sağlar. Her bir kaynak, kendi özel kilidine sahiptir ve özel erişim yönetimi için kullanılabilir. Dağıtılmış Kilit API'si, kaynaklara tutarlı erişimi sağlayarak veri tutarlılığını ve bütünlüğünü korur. Örneğin, aynı anda birden fazla uygulama örneğinin aynı veriyi güncellemesi gerektiğinde, kilitler veri tutarlılığını sağlar. Dapr 1.8 ile alfa durumunda tanıtılmıştır.

10. Kriptografi (Cryptography)

Dapr'ın Şifreleme API'si, güvenli iletişim gerektiren uygulamalar için önemli bir yapı taşı sağlar. Şifreleme işlemleri, verilerin güvenliğini sağlamak ve gizliliğini korumak için kullanılır.

Dapr Mimarisi

Dapr'ın çalışma mantığının temelinde yan bileşen deseni yatmaktadır. Genel olarak açıklamak gerekirse; Dapr ana uygulama kodumuzda herhangi bir değişiklik yapmak yerine, kendini uygulamayla konuşan uygulamanın dış dünyaya açılan kapısı olarak konumlandırır.

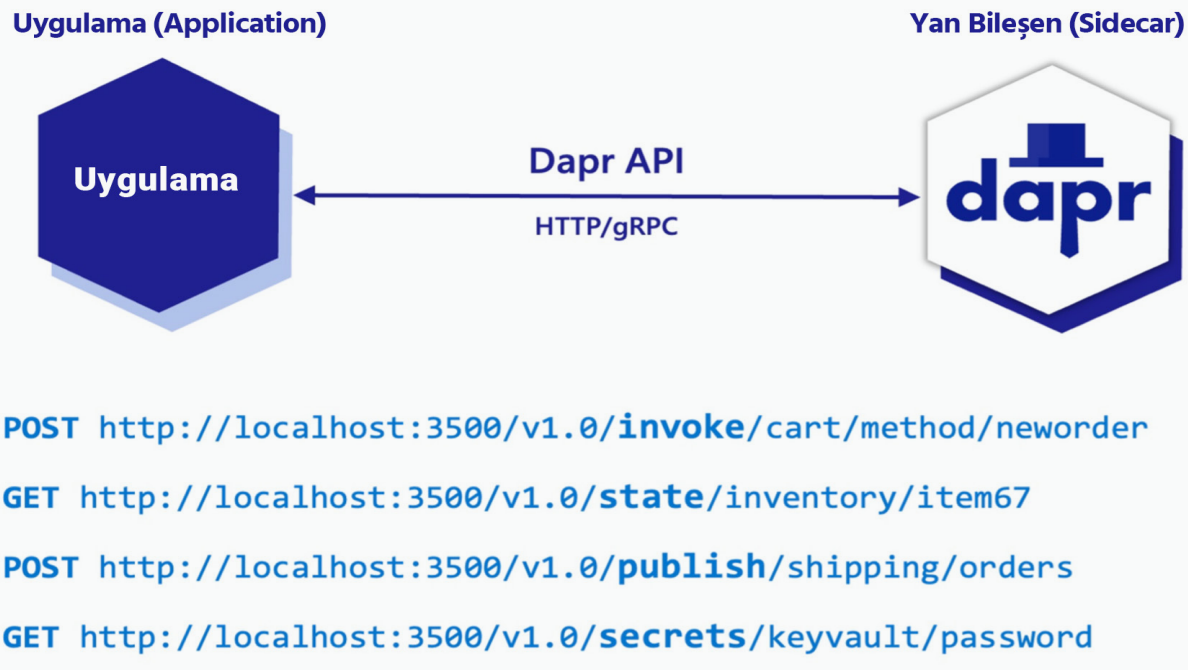
Dapr, çeşitli araçlar ve bileşenlerle mikroservis mimarisinin zorluklarını hafifletmeyi amaçlar. Bu araç ve bileşenler şu şekilde sıralanabilir.

- **Dapr Komut Satırı Arayüzü (CLI):** Komut satırı aracı Dapr ile etkileşim kurmak için kullanılır. CLI sayesinde Dapr ortamı kurulabilir ve yapılandırılabilir. Ayrıca, Dapr uygulaması yönetebilir ve hata ayıklamada kullanılır.
- **Dapr Helm Şablonları (Helm Charts):** Dapr Helm Şablonları, Dapr'ın Kubernetes üzerinde kolayca kurulabilmesi ve yönetilebilmesi için kullanılır. Helm Şablonları ile Dapr bileşenlerinin güncellemeleri ve yapılandırmaları yönetilebilir.
- **Dapr Arayüzü:** API bir uygulamanın Dapr çalışma zamanı ile nasıl etkileşim kuracağını tanımlar. Servisler arası çağrılar yapılabilir ve bu çağrıları yönetilebilir. API, Dapr yapı taşlarını kullanmasını sağlar.
- **Dapr Çalışma Zamanı:** Dapr'ın çekirdeğini oluşturur ve uygulamanın çalıştığı ortamda mikroservislerin iletişimini yönetir. Go dilinde geliştirilmiştir. Dapr bileşenlerinin işlevselliğini sağlar. Mikroservisler arasındaki ağ trafiğini yönlendirir.

- **Dapr Sunucusu:** Dapr Host, Dapr bileşenlerinin çalıştığı sunucu veya ortamdır. Dapr bileşenleri arasında iletişim sağlanır ve yönetilir. Bağımsız bir işlem olarak çalışırken, Kubernetes'te uygulamanızın pod'unda bir yan bileşen konteyner olarak çalışır.
- **Dapr Operatörü:** Kubernetes üzerinde Dapr bileşenlerini yöneten bir denetleyicidir. Bağlantıları ve yapılandırmaları yönetir. Dapr bileşenlerinin kurulumunu ve yapılandırmasını otomatikleştirir.
- **Dapr Yan Bileşen Enjektörü:** Kubernetes üzerinde Dapr yan bileşen konteynerlerinin otomatik olarak enjekte edilmesini sağlar. Kubernetes pod'larına Dapr yan bileşenlerini ekler. Yan bileşen enjekte edilen pod'ların doğru yapılandırmalarla çalışmasını sağlar.
- **Dapr Yerleştirme Servisi (Placement Services):** Aktör tabanlı uygulamaların dağıtımını yönetir. Dapr pod'ları arasında Aktör örneklerini dağıtma (veya yerleştirme) amacına sahiptir. Aktörlerin hangi düğümde çalışacağını belirler ve ölçeklenebilirliğini sağlar.
- **Dapr Takibi:** Dapr bileşenleri arasındaki güvenli iletişimi sağlar. Dapr tarafından kullanılan sertifikaları sağlamak ve yönetmek için yerleşik bir Sertifika Otoritesi (CA) olup, şeffaf bir karşılıklı Taşıma Katmanı Güvenliği (mTLS) sağlar. Bileşenler arası güvenli bağlantıların kurulmasını sağlar.

Dapr'ın yan bileşen işleminin adı 'daprd' dir. Dapr, daprd işlemini gerçekleştirebilmek için şunları sunar:

- Uygulama içinden erişilen uygulamanın altyapısal bağımlılıklarına karşılık gelecek yapı birimlerinin yönetilmesini sağlayan Building block API (Yapı taşı API),
- Yan bileşen işlemlerinin tanımlarını içeren bir meta veri API,
- Yan bileşenlerin sağlık durumunu ve hazır olup olmadığını sağlayan bir health API.

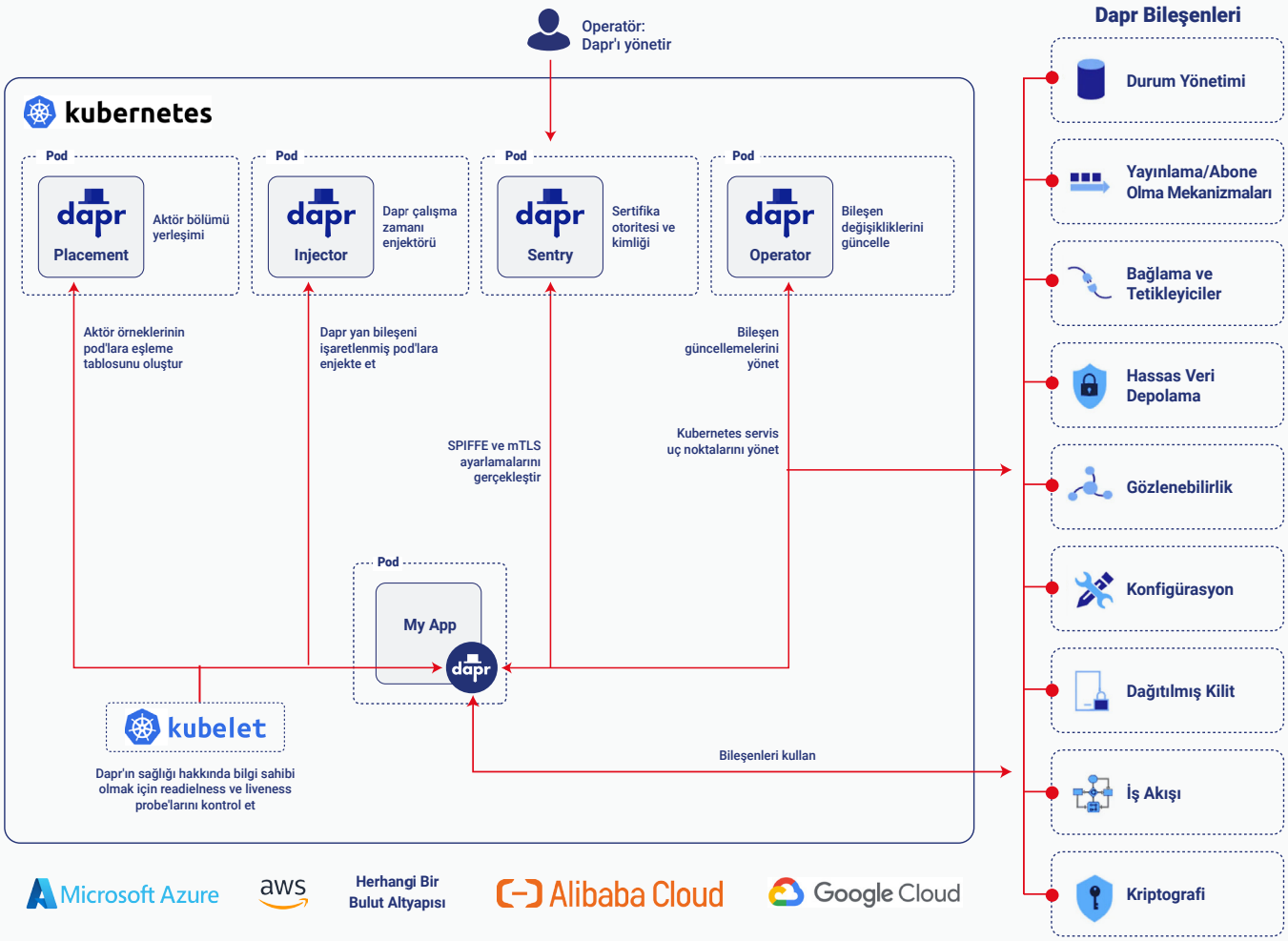


Şekil 9. Uygulamanın Dapr'a Erişimi

Şekil 9, Dapr'ın state management, publish, secrets ve HTTP POST/GET metodlarını nasıl desteklediğini göstermektedir. Ana uygulama, Dapr yerine yan bileşeni ile, gRPC veya HTTP protokolleriyle iletişime geçer. HTTP üzerinden istek atılmak istendiğinde Dapr sadece POST ve GET metotlarını desteklemektedir. Bu metotlar kullanılarak tetiklenmek istenen Dapr API'leri çağrılabilir. Uygulama, gerçekleştirmek istediği altyapısal operasyonları, Dapr'da mevcut olan yapı birimlerine karşılık gelen yan bileşen üzerinden tetikler.

Bileşen Tanımı

Dapr'da bir yapı bloğu/bileşen kullanılmak istenirse, öncelikle bu bileşenin tanımı yapılmalıdır. Bu bileşen tanımının formatı Kubernetes'de (Bknz. https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/) görülen özel kaynak tanımı (CRD) yapısıyla aynıdır. Her bileşen tipinin kendine özgü tanımları vardır ve belirli bir formata uymalıdır. Bu format, bileşenin işlevselliğini, bağımlılıklarını, yapılandırmasını ve diğer önemli özelliklerini içerir. Böylece bileşenler Kubernetes ve Dapr ortamında uyumlu ve etkili bir şekilde çalışabilir. Bir bileşenin yapılandırma dosyası, yapılandırma ayarları, bağımlılıklar, veri ve kaynak yönetimi, güvenlik ve erişim kontrolleri ile performans ve izleme bilgilerini içermelidir.



Şekil 10. Kubernetes Ortamında Dapr

Kubernetes ortamında Dapr (Distributed Application Runtime), mikroservislerin geliştirilmesi ve yönetilmesi için güçlü bir araçtır. Dapr, Kubernetes ile entegre olarak, mikroservislerin dağıtılmış yapısını basitleştiren ve ölçeklenebilirliğini artıran çeşitli yapı taşları sunar. Bu yapı taşları arasında durum yönetimi, servis çağırma, yayınlama ve abone olma, bağlamalar ve gözlemlenebilirlik gibi özellikler bulunur. Kubernetes ortamında Dapr kullanmanın en önemli avantajlarından biri, mikroservislerin birbirleriyle etkileşimlerini kolaylaştıran bir dizi API sağlamasıdır. Dapr, Kubernetes'in sağladığı temel kaynakları genişleterek, servisler arası iletişim, veri yönetimi ve mesajlaşma gibi işlevleri basitleştirir. Örneğin, Dapr'ın servis çağırma API'si, mikroservislerin birbirine güvenli ve verimli bir şekilde HTTP veya gRPC üzerinden çağrı yapmasını sağlar, bu da mikroservisler arasında sorunsuz bir iletişim sağlar.

Ayrıca, Dapr'ın yayınlama ve abone olma mekanizması, mikroservisler arasında asenkron mesajlaşmayı destekler. Bu özellik, yüksek ölçeklenebilirlik ve düşük gecikme süreleri ile veri akışlarını yönetmeyi kolaylaştırır. Kubernetes'in otomatik ölçeklenebilirlik ve yüksek erişilebilirlik özellikleri ile birleştiğinde, Dapr bu işlevsellikleri daha da güçlendirir ve mikroservislerin dinamik olarak ölçeklenmesini sağlar.

Dapr, aynı zamanda Kubernetes ile birlikte gözlemlenebilirliği artırır. Gözlemlenebilirlik API'leri, uygulama performansını izlemeye ve sorunları hızlı bir şekilde tespit etmeye olanak tanır. Bu, Dapr'ın sağlık kontrolleri ve hata ayıklama süreçlerinde önemli bir rol oynar. Kubernetes'in merkezi yönetim özellikleri ile birleşerek, Dapr'ın sağladığı gözlemlenebilirlik, sistemin genel performansını ve güvenilirliğini artırır.

```
apiVersion: dapr.io/v1alpha
kind: Component
auth:
  secretstore: [ SECRET-STORE-NAME ]
metadata:
  name: [ COMPONENT-NAME ]
  namespace: [ COMPONENT-NAMESPACE ]
spec:
type: [ COMPONENT-TYPE ]
  version: v1
  initTimeout: [ TIMEOUT-DURATION ]
  ignoreErrors: [ BOOLEAN ]
  metadata:
    - name: [ METADATA-NAME ]
      value: [ METADATA-VALUE]
scopes:
  - [ APPID ]
  - [ APPID ]
```

Şablonda tanımlı olan alanlar ve bu alanların ne için kullanıldığı bilgisi şu şekilde verilmektedir:

apiVersion: Dapr'ın versiyonunun tanımlandığı zorunlu bir alandır.

kind: Oluşturulmak istenen kaynağın tipini tanımlar zorunlu bir alandır ve bileşen tipindeki kaynaklar için "Component" olmak zorundadır.

auth: Bileşenlerin ihtiyacı olduğu kullanıcı adı şifre gibi bilgileri doğrudan konfigürasyon dosyalarına yazmak güvenlik açığı oluşturabilir. Bu açığı engellemek için Kubernetes'den tanıdığımız gizli anahtar depoları (secret store) kullanılır. Auth alanında da bu secret store'un adı tanımlanır. Zorunlu olmayan bir alandır.

metadata: Metadata alanı bileşenin tanım bilgilerini içeren ad (name) ve ad alanı (namespace) alanlarını içerir.

metadata.name: Kullanılmak istenen kaynağın adının tanımlandığı zorunlu bir alandır.

metadata.namespace: Kullanılmak istenen kaynağın hangi ad alanı (namespace) içerisinde bulunacağının tanımının yapıldığı zorunlu olmayan bir alandır.

spec: Kullanılmak istenen kaynağa ait tanımlamaları içeren alandır.

spec.type: Kullanılmak istenen kaynağın tipini içeren zorunlu bir alandır. Bu alan için genel olarak "anaTip"."araçAdı" gibi bir format kullanılmaktadır. Örneğin; "state.redis", "pubsub.kafka".

spec.version: Kullanılmak istenen kaynağın versiyon bilgisinin yönetildiği zorunlu bir alandır.

spec.ignoreErrors: Kullanılmak istenen kaynakta alınan hataların Dapr'ın yan bileşeni ayağa kaldırmasına engel olup olmamasının tanımlandığı zorunlu olmayan bir alandır. Ön kabul edilen değer false'tur. Kaynakta alınan hatalar yan bileşenin ayağa kalkmasını engelleyecek şekilde tanımlanmaktadır.

spec.initTimeout: Kullanılmak istenen kaynağın hazır olması için beklenecek sürenin tanımlandığı zorunlu olmayan bir alandır. Ön tanımlı değeri 5 sn'dir.

spec.metadata: Kullanılmak istenen kaynağa özgü ayarlamaların/tanımlamaların yapıldığı alandır. Kullanıcı adı, şifre, erişileceği url vb. gibi tanımlar yapılmaktadır. Spring uygulamalarındaki application.properties'de yapılan tanımlara benzer tanımlamalar yapılmaktadır.

scopes: Tanımlanan bileşenin hangi uygulamalardan erişilebileceğini kısıtlamak istediğimizde bu alan kullanılmaktadır. Erişilmek istenen uygulamaların appId'leri bu alanda sıralanmaktadır.

Ön Yüklü Gelen ve Eklenebilir Bileşenler

Dapr'ın oldukça zengin bir bileşen kütüphanesi vardır. Bu bileşenler Dapr'ın ana reposunda değil de ek bir repo'da tutulmaktadır ve onlar da aynı Dapr gibi açık kaynak olarak geliştirilmektedir. Daha sık sürüm alınabilmesi (release) için böyle bir ayrıma gidilmiştir. Bu sayede yeni bileşen ekleneceği zaman ana repo'da ana sürümü beklemeden güncellemeler çıkılabilmektedir.

Bu bileşenlere ek olarak kullanıcılar kendi özel isteklerine göre de bileşen oluşturabilmektedir. Buna Dapr ekosisteminde eklenebilir (pluggable) bileşen denmektedir. Bu bileşenler Dapr'ın building block API'sini kullanacak şekilde oluşturulur. Bu işlemi gerçekleştirmek için aşağıdaki adımlar izlenir:

- Birinci olarak hangi tipte eklenebilir bileşenin oluşturulmak istendiğine karar verilir, Dapr'ın v.13'üne göre State store, pub/sub, bindings ve secret store'lar için eklenebilir bileşen tanımlanabilmektedir.

- İkinci olarak tipe ait proto tanım dosyası bulunur. Bu proto dosyası oluşturulmak istenen eklenebilir bileşenin implement etmesi gereken fonksiyonların tanımlarını içeren bir arayüzdür. Örnek proto dosyasının arayüzü tanımlayan kısmı şu şekilde görünmektedir:

```
service PubSub {
  // Verilen meta veriler ile pubsub bileşenini başlatır.
  rpc Init(PubSubInitRequest) returns (PubSubInitResponse) {}

  // Uygulanan pubsub özelliklerinin bir listesini döndürür.
  rpc Features(FeaturesRequest) returns (FeaturesResponse) {}

  // Verilen konu için yeni bir mesaj yayınlar.
  rpc Publish(PublishRequest) returns (PublishResponse) {}

  rpc BulkPublish(BulkPublishRequest) returns (BulkPublishResponse) {}

// Sunucu (PubSub bileşeni) ile bir akış kurar; sunucu mesajları
// istemciye (daprd) gönderir. İstemci onayları sunucuya geri gönderir.
// Sunucu, herhangi bir hata durumunda akışı kapatacak ve durumu döndürecektir.
// Bağlantı kapalı olduğunda, istemcinin akışı yeniden kurması gerekir.
// İlk mesaj, tüm akış çekişi için kullanılacak bir `topic` özelliği
içermelidir.
  rpc PullMessages(stream PullMessagesRequest)
      returns (stream PullMessagesResponse) {}

  // Ping the pubsub. Used for liveness porpuses.
  rpc Ping(PingRequest) returns (PingResponse) {}
}
```

Bu proto dosyasına göre pubsub tipinde bir eklenebilir bileşen oluşturulmak isteniyorsa; init, features, publish, pullMessages ve ping fonksiyonlarının tanımlanması gerekmektedir. Ana uygulama ile bileşenler daha önce de bahsedildiği gibi gRPC kullanarak iletişime geçtiği için eklenmek istenilen bileşen için de temelde tanımlanmış proto dosyasına göre bir gRPC servisi oluşturması beklenmektedir.

Bu işlem için proto dosyalarının kullanılmasının diğer bir pozitif yanı ise protocol buffers ve gRPC'nin kapsamlı araç desteği ile beraber, birçok dil için kod üretimini sağlamasıdır. Bu sayede çalıştırılacak küçük bir komut satırıyla oluşturulmak istenen bileşen için taslak proje oluşacaktır. Örneğin aşağıdaki komut satırı çalıştırıldığında Go dilinde pubsub eklenebilir bileşeni için taslak proje oluşacaktır.

```
protoc --go_out=. --go_opt=paths=source_relative \\    --go-grpc_out=. --go-
grpc_opt=paths=source_relative \\    pubsub/pubsub.proto
```

Bu taslak proje, istek ve yanıtlar için kullanılacak sınıflar, gRPC kullanarak tetiklenecek fonksiyon şablonunu içerir. Oluşturulmak istenen bileşen için de bu şablonun içinin doldurulması beklenmektedir.

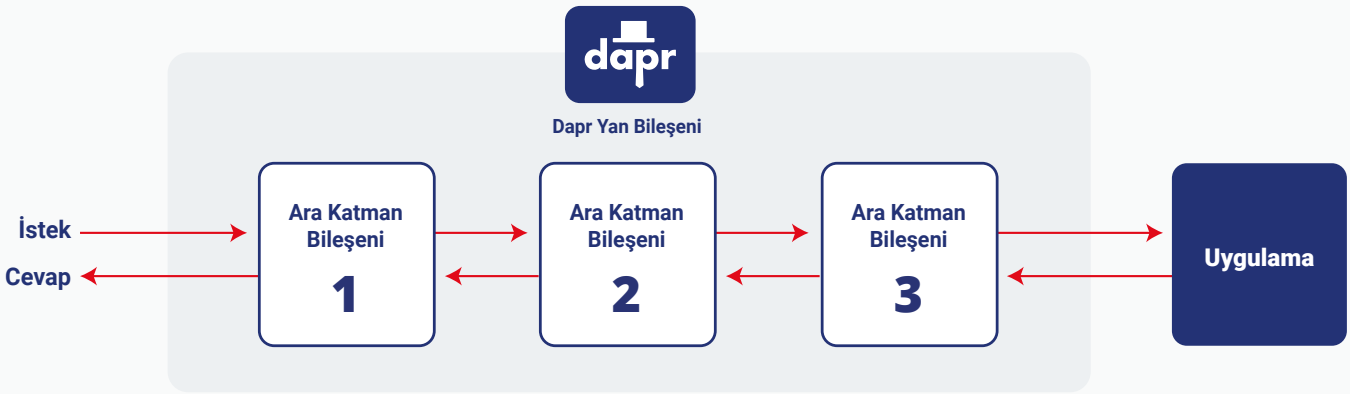
```
func (r *redisStreams) Publish(ctx context.Context, req *pubsub.PublishRequest)
error {
    if r.closed.Load() {
       return errors.New("component is closed")
    }

    redisPayload := map[string]interface{}{"data": req.Data}

    if req.Metadata != nil {
       serializedMetadata, err := json.Marshal(req.Metadata)
       if err != nil {
          return err
       }
       redisPayload["metadata"] = serializedMetadata
    }
    _, err := r.client.XAdd(ctx, req.Topic, r.clientSettings.MaxLenApprox,
redisPayload)

    if err != nil {
       return fmt.Errorf("redis streams: error from publish: %s", err)
    }
    return nil
}
```

Redis için oluşturulmuş pubsub eklenebilir bileşeninin publish metodunu Go dilindeki örneğini yukarıda görebilirsiniz. Publish herhangi bir sonuç döndürmediği için sadece hata (error) dönmektedir. Fonksiyonun parametrelerinde ise proto dosyasında tanımlı isteğin olduğu görülmektedir.

Ara Katman Bileşeni

Her Dapr yan bileşeni ayağa kalkarken 2 ara katman bileşeniyle beraber kalkar. Ön tanımlı olarak bunlar tracing ve CORS ara katmanlarıdır. Dapr'ın esnek yapısı sayesinde ara katman tanımlama fırsatı sağlar. Bu ara katmanlar ana bileşene gelen isteklere tanımlanabildiği gibi, ana bileşenden çıkan cevaplara da tanımlanabilmektedir. Ara katman denildiğinde sadece WEB istekleri düşünülebilir, ama Dapr'ın sunduğu her API'ye (durum değişimleri, pub/sub, secret yönetimi vb.) bu ara katmanlar eklenebilir. Ara katmanların birbiri ardına eklemede de herhangi bir kısıt bulunmamaktadır. Ara katman zincirleri oluşturulup kullanılabilir.

Dapr'ın uygulama ile dış dünya arasındaki kapı olduğu daha önce belirtilmiştir. Uygulamaya erişmek isteyen tüm istekler, Dapr yan bileşeni aracılığıyla uygulamaya yönlendirilmektedir. Bu başlıkta bahsedilen ara katmanlar, yan bileşenin içerisinde tanımlanmaktadır. **İsteğin, uygulamaya erişene kadar geçmesi gereken aşamalar Şekil 11'deki diyagramda gösterilmektedir.**



Şekil 11. Dapr HTTP API'leri: Ara Katman Bileşenleri ve İşleyişi

Ara katmanların uygulamaya erişirken mi yoksa uygulamaya eriştikten sonra mı çalışması gerektiği ise ara katman tanımı yapılırken belirtilmektedir. Bu zincirlerin tanımı ise konfigürasyon (configuration) tipinde bir bileşen oluşturup onun içinde yapılır;

```
apiVersion: dapr.io/v1alpha1
kind: Configuration
metadata:
  name: pipeline
  namespace: default
spec:
  httpPipeline:
    handlers:
      - name: oauth2
        type: middleware.http.oauth2
      - name: uppercase
        type: middleware.http.uppercase
```

Bu tanıma göre önce her istek oauth2 üzerinden geçerek yetki kontrolleri yapılır, daha sonrasında istek uppercase bileşenine girip büyük harfe dönüştürülür. Ara katman yazılmak istenirse Go dilinin kullanılması gerekmektedir. Dapr tamamen Go dilinde yazılmıştır, bu nedenle Dapr bileşenleri geliştirilirken Go dilinin kullanılması gerekmektedir. Bu durum, uygulamanın da Go dilinde yazılması gerektiği izlenimini yaratabilir. Ancak, bu sadece Dapr'a doğrudan müdahaleyi ve geliştirme sürecini kapsamaktadır. Uygulama herhangi bir dilde yazılabilir; çünkü Dapr, HTTP veya gRPC protokollerini kullanarak dil bağımsız olarak haberleşmektedir. Dolayısıyla, uygulama dili, Dapr ile etkileşimde kısıtlama oluşturmamaktadır.

Ara katman yazmak için önce middleware arayüzünün implement edilmesi gerekmektedir. Middleware arayüzü şu şekildedir:

```
type Middleware interface {
    GetHandler(metadata middleware.Metadata) (func(next http.Handler) http.
Handler, error)
}
```

Bu arayüzü implement eden örnek fonksiyon şu şekilde tanımlanabilir:

```
func (m *customMiddleware) GetHandler(metadata middleware.Metadata) (func(next
http.Handler) http.Handler, error) {
    var err error
    return func(next http.Handler) http.Handler {
       return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
          // İstek işlenmeden önce çalışmasını istediğimiz
          // işlemler bu aşamada gerçekleştirilir.

          // Ara katmandan sonra gidilecek nokta bu anda çağrılır
          next.ServeHTTP(w, r)

          // İstek işlendikten sonra çalışmasını istediğimiz
          // işlemler bu aşamada gerçekleştirilir.
       }
    }, err
}
```

Kullanım yöntemi, programların belirli bir işlevselliği merkezi bir noktada toplamasını sağlayan AOP (Aspect Oriented Programming) mantığına oldukça benzerdir. Temelde aynı görevi gerçekleştirirler.

Sonuç ve Öneriler

Dapr, dağıtılmış mikroservis uygulamalarının karmaşıklığını azaltmayı hedefleyen bir uygulama çalışma zamanıdır. Geliştiricilere, API'ler aracılığıyla temel dağıtılmış uygulama yetenekleri sunarak esnek ve dayanıklı mikroservisler oluşturma imkânı tanır. Dapr'ın sunduğu yapı taşları arasında durum yönetimi, hizmetten hizmete çağrı, pub/sub mesajlaşma, gizli dizi yönetimi ve gözlemlenebilirlik gibi önemli işlevler yer almaktadır. Herhangi bir programlama dili veya çerçeve ile uyumlu olması ve geniş dağıtım seçenekleri sunması, modern uygulama geliştirme süreçlerini daha verimli hale getirmektedir.

Literatürdeki çalışmalar mikroservislerin yönetimi ve servis mesh mimarileri üzerine yoğunlaşmış olsa da, Dapr gibi uygulama çalışma zamanları hakkında çok fazla çalışmaya rastlanmamıştır. Bu bağlamda, Dapr'ın potansiyel faydaları, avantajları, dezavantajları ve kullanım senaryoları üzerine daha fazla araştırma yapılması gerekmektedir. Bu çalışma, kaynaklardaki bu boşluğu doldurmayı ve Dapr'ın mikroservis mimarilerinde nasıl etkin bir şekilde kullanılabileceğini açıklamayı amaçlamaktadır.

Nitel bir araştırma tasarımı kullanılarak gerçekleştirilen bu çalışma, Dapr'ın teorik altyapısını ve pratik uygulama örneklerini incelemiştir. Kaynak taraması, Dapr dokümantasyonu ve gerçek dünya uygulamalarından elde edilen veriler kullanılarak, Dapr'ın bileşenleri, işlevleri ve entegrasyon noktaları detaylı bir şekilde analiz edilmiştir. Çalışma, Dapr'ın mikroservis tabanlı uygulamalarda nasıl kullanılabileceği, sağladığı avantajlar ve potansiyel kullanım senaryoları üzerine odaklanarak, geliştiricilere yol göstermeyi amaçlamaktadır.

Sonuç olarak bu çalışmada, Dapr'ın dağıtılmış sistemlerde güvenli, ölçeklenebilir ve yönetilebilir uygulamalar geliştirme sürecini nasıl desteklediği ve geliştiricilere nasıl katkı sağladığı detaylı bir şekilde ortaya konmuştur. Dapr'ın modern uygulama geliştirme süreçlerindeki potansiyel rolü, ileriye dönük araştırmalar ve uygulamalar için önemli bir temel oluşturmaktadır. Bu nedenle Dapr'ın daha geniş bir şekilde araştırılması ve çeşitli kullanım senaryolarında test edilmesi önerilmektedir.

Kaynakça

1. Devoteam. (n.d.). **Review of Dapr.** Retrieved August 23, 2024, from https://www.devoteam.com/expert-view/review-dapr/#:~=In%20essence%2C%20Dapr%20helps%20reduce,-sub%20messaging%2C%20secrets%20management%2C%20and

2. Bajaj, G. (2021). **Demystifying Dapr: State Management Block for Distributed Apps in Development Environment.** Retrieved August 23, 2024, from https://gaganbajaj.medium.com/demystifying-dapr-state-management-block-for-distributed-apps-in-development-environment-2d6335e8ff61

3. Dapr. (n.d.). **Dapr Official Website.** Retrieved August 23, 2024, from https://dapr.io/

4. Sims, J. (2021). **Simplifying Microservices with Aspire: ASPIR8 & Dapr - The Radius Platform.** Retrieved August 23, 2024, from https://medium.com/@josephsims1/simplifying-microservices-with-aspire-aspir8-dapr-the-radius-platform-6dabb6b25ec1

5. Prakash, R. (2021). **What is Dapr?.** Retrieved August 23, 2024, from https://medium.com/@raviprakash_29609/what-is-dapr-196fc8c565f

6. Dapr Documentation. (n.d.). **Building Blocks Concept.** Retrieved August 23, 2024, from https://docs.dapr.io/concepts/building-blocks-concept/

7. Microsoft. (2019). **Announcing Dapr: Open Source Project to Build Microservice Applications.** Retrieved August 23, 2024, from https://cloudblogs.microsoft.com/opensource/2019/10/16/announcing-dapr-open-source-project-build-microservice-applications/

8. Stackademic. (2021). **Unlocking Distributed Systems: A Deep Dive into Dapr's Building Blocks.** Retrieved August 23, 2024, from https://blog.stackademic.com/unlocking-distributed-systems-a-deep-dive-into-daprs-building-blocks-1f4df5d22f8b

9. O'Reilly Media. (2021). **Practical Microservices with Dapr.** Retrieved August 23, 2024, from https://learning.oreilly.com/library/view/practical-microservices-with/9781803248127/

10. O'Reilly Media. (2021). **Learning Dapr.** Retrieved August 23, 2024, from https://www.oreilly.com/library/view/learning-dapr/9781492072416/introduction01.html#language_support

11. Dapr Documentation. (n.d.). **Overview.** Retrieved August 23, 2024, from https://docs.dapr.io/concepts/overview/

12. Dapr Documentation. (n.d.). **Available Component Types.** Retrieved August 23, 2024, from https://docs.dapr.io/concepts/components-concept/#available-component-types

13. Mstryoda. (2021). **Distributed Application Runtime: Dapr'a Genel Bakış.** Retrieved August 23, 2024, from https://mstryoda.medium.com/distributed-application-runtime-dapra-genel-bakış-d2677b85b0f7

14. Dapr Documentation. (n.d.). **Pluggable Components Overview.** Retrieved August 23, 2024, from https://docs.dapr.io/developing-applications/develop-components/pluggable-components/pluggable-components-overview/

15. Dapr Documentation. (n.d.). **Component Schema.** Retrieved August 23, 2024, from https://docs.dapr.io/reference/resource-specs/component-schema/

16. Dapr Documentation. (n.d.). **Middleware Components.** Retrieved August 23, 2024, from https://docs.dapr.io/operations/components/middleware/

17. Dapr Documentation. (n.d.). **Configuration Overview.** Retrieved August 23, 2024, from https://docs.dapr.io/operations/configuration/configuration-overview/

18. Dapr Documentation. (n.d.). **Pluggable Components Registration.** Retrieved August 23, 2024, from https://docs.dapr.io/operations/components/pluggable-components-registration/

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)