Yüksek Performanslı Gerçek Zamanlı PHP Uygulamaları için Süreçler Arası İletişim Mimarilerinin Analizi ve Uygulaması

Bölüm 1: Mimari Zorluğun Dekonstrüksiyonu

Modern, gerçek zamanlı PHP uygulamaları geliştirmek, geleneksel PHP paradigmasının ötesine geçen mimari kararlar gerektirir. Kullanıcının karşılaştığı ext-zmq eklentisi kurulum hatası, sadece teknik bir engel değil, aynı zamanda daha derin bir mimari sorunun belirtisidir: durum bilgisi olmayan (stateless) HTTP dünyası ile durum bilgisi olan (stateful) WebSocket dünyası arasında köprü kurma ihtiyacı. Bu rapor, bu temel zorluğu analiz edecek, potansiyel çözümleri değerlendirecek ve sağlam, performanslı ve ölçeklenebilir bir sistem için en uygun yolu detaylandıracaktır.

1.1. Modern PHP İkilemi: Durumsuz HTTP ve Durumlu WebSockets

Çağdaş ve interaktif PHP uygulamaları, genellikle iki farklı çalışma modelini bir arada barındıran "bölünmüş beyin" mimarisiyle çalışır. Bu iki modelin doğasını anlamak, aralarındaki iletişim ihtiyacını kavramak için temel teşkil eder.

- **HTTP İstek İşleyicisi (örneğin, PHP-FPM):** Bu bileşen, PHP'nin geleneksel çalışma modelini temsil eder. Her HTTP isteği için yeni bir süreç başlatılır, bu süreç kısa ömürlüdür ve durumsuzdur. Görevi, geleneksel web isteklerini almak, veritabanıyla etkileşime geçmek, iş mantığını işlemek ve bir HTTP yanıtı gönderdikten sonra sona ermektir.[1] Bu model, web'in temel istek-yanıt doğası için son derece verimli olsa da, sunucudan istemciye anlık veri gönderme yeteneğinden yoksundur.
- **WebSocket Sunucusu (örneğin, Ratchet):** Bu bileşen, modern PHP'nin evrimini temsil eder. Komut Satırı Arayüzü (CLI) aracılığıyla başlatılan, uzun ömürlü ve durum bilgisi olan bir süreçtir.[2] Birden çok istemciyle kalıcı, çift yönlü bağlantılar kurar ve bu bağlantıların durumunu (örneğin, bağlantı nesneleri, kullanıcı kimlikleri)

kendi belleğinde tutar. Bu, her istekten sonra sona eren geleneksel PHP yürütme modelinden temel bir sapmadır.[2]

- **İletişim Uçurumu:** Temel sorun tam olarak bu noktada ortaya çıkar. Durumsuz HTTP dünyasında başlatılan bir eylem – örneğin, bir mobil uygulamanın bir QR kodunu tarayıp onayladığını bir POST isteği ile sunucuya bildirmesi – durum bilgisi olan WebSocket dünyasında bir tepkiyi tetiklemelidir – örneğin, sunucunun belirli bir web tarayıcısına "giriş başarılı" mesajı göndermesi. Bu iki ayrı ve farklı yaşam döngüsüne sahip süreç arasında güvenilir bir şekilde veri aktarmak, sağlam bir Süreçler Arası İletişim (Inter-Process Communication - IPC) mekanizması gerektirir.

1.2. IPC Uçurumunu Kapatmak için Aday Çözümler

Bu mimari zorluğu çözmek için birkaç potansiyel teknoloji bulunmaktadır. Bu rapor, kullanıcının hedefleri ve karşılaştığı sorunlar bağlamında üç ana adayı analiz edecektir.

- **Aday A: Yüksek Performanslı Mesajlaşma Kütüphanesi (ZeroMQ):** Kullanıcının ilk tercihi olan bu çözüm, süreçler arasında doğrudan, yüksek hızlı ve düşük gecikmeli iletişim sağlamak için tasarlanmış "aracısız" (brokerless) bir kütüphanedir.[4] Her sürecin diğerine doğrudan bağlanmasını sağlar.
- **Aday B: Merkezi Bellek İçi Aracı (Redis Pub/Sub):** Bu modelde, süreçler birbirleriyle doğrudan konuşmak yerine, merkezi bir sunucu aracılığıyla dolaylı olarak iletişim kurar. Bu yaklaşım, yayıncıyı (publisher) aboneden (subscriber) ayırarak mimariyi basitleştirir.[6] Kullanıcının kendi araştırma dokümanında da bu model, ölçeklenebilir bir çözüm olarak önerilmektedir.[2]
- **Aday C: Veritabanını Mesaj Kuyruğu Olarak Kullanmak:** Kullanıcının acil durum planı olan bu yaklaşım, yeni bir teknoloji eklemeden mevcut altyapıyı (MySQL) kullanarak bir çözüm sunar. Ancak bu, veritabanını tasarlanmadığı bir görev için kullanmanın getirdiği önemli performans ve karmaşıklık sorunlarını beraberinde getirir.[8]

Kullanıcının karşılaştığı ext-zmq kurulum hatası, bu nedenle basit bir yazılım yükleme sorunundan çok daha fazlasıdır. Bu hata, projenin temel taşlarından biri olan IPC mekanizması üzerine yeniden düşünmek için bir katalizör görevi görmektedir. "Bu kurulumu nasıl düzeltirim?" sorusundan ziyade, "Sağlam ve performanslı bir sistem hedefiyle bu iletişimi sağlamanın *en iyi* yolu nedir?" şeklindeki daha stratejik soruyu sormayı gerektirir. Bu rapor, bu ikinci ve daha değerli soruya kapsamlı bir yanıt

sunacaktır.

Bölüm 2: Windows için PHP 8.4 Üzerinde ext-zmq Uygulamasının Analizi

Kullanıcının karşılaştığı temel sorun, ext-zmq eklentisini Windows üzerindeki modern PHP yığınına kurma girişimidir. Bu bölüm, bu görevin neden pratik olmadığını ve neden stratejik bir değişiklik gerektiğini teknik kanıtlarla ortaya koyacaktır.

2.1. Modern Windows PHP için PECL DLL Manzarası

Windows üzerinde bir PHP eklentisi kurmanın en yaygın ve tercih edilen yolu, önceden derlenmiş bir DLL (Dynamic Link Library) dosyası kullanmaktır.[10] Bu DLL'ler genellikle PHP'nin resmi eklenti deposu olan PECL (PHP Extension Community Library) üzerinden temin edilir.

Ancak, pecl.php.net adresindeki ZMQ paketi sayfası incelendiğinde, önceden derlenmiş DLL'lerin en son 2016 yılında, PHP 7.x sürümleri için yayınlandığı görülmektedir.[11] Kullanıcının son derece spesifik olan

PHP 8.4.10 (ZTS Visual C++ 2022 x64) yapısı için resmi olarak derlenmiş bir DLL mevcut değildir. Bu durum, yalnızca ZMQ'ya özgü bir sorun değildir. PHP for Windows ekibinin de belirttiği gibi, PECL DLL derleme altyapısı genel olarak kesintiler yaşamış ve yeni bir CI sürecine geçiş aşamasındadır.[13] Bu, en yeni PHP sürümleri için güvenilir ve güncel eklenti DLL'leri bulmanın genel bir zorluk olduğunu doğrulamaktadır.

2.2. Manuel Kurulum ve Derlemenin İncelikleri

Resmi bir DLL olmadığında, geliştiriciler genellikle manuel kurulum veya kaynaktan derleme yollarına başvurur. Ancak ext-zmq için bu yollar son derece karmaşık ve risklidir.

- **Bağımlılık Cehennemi:** php_zmq.dll tek başına çalışan bir dosya değildir. Çalışabilmesi için temel ZeroMQ kütüphanesi olan libzmq.dll'e ve modern sürümler için genellikle kriptografi kütüphanesi olan libsodium.dll'e kritik bağımlılıkları vardır.[14] Bu bağımlılıkların her birinin, PHP yapısıyla (sürüm, iş parçacığı güvenliği, derleyici, mimari) tam olarak uyumlu olması gerekir.[10]
- **"DLL Karıştırmacası":** Başarılı bir kurulum, bu farklı DLL dosyalarının doğru dizinlere yerleştirilmesini gerektirir. Genellikle php_zmq.dll dosyası PHP'nin ext klasörüne konulurken, bağımlılığı olan libzmq.dll gibi dosyaların sistemin PATH değişkeni tarafından bulunabilmesi için PHP'nin kök dizinine, hatta bazen Apache'nin bin dizinine kopyalanması gerekir.[10] Bu süreç, deneme yanılmaya dayalı, kırılgan ve hataya açık bir yapılandırma yaratır.
- **Derleme Cenderesi:** Önceden derlenmiş ve uyumlu bir DLL bulunamadığında geriye kalan tek seçenek, eklentiyi kaynaktan derlemektir. Bu, sıradan bir PHP geliştiricisi için pratik olmayan, uzmanlık gerektiren bir görevdir. Kullanıcının PHP yapısına uygun bir derleme için belirli bir geliştirme ortamı gerekir: PHP 8.4, VS2022 ile derlendiği için Visual Studio 2022, Windows için PHP SDK'sı, libzmq kaynak kodu ve php-zmq eklentisinin kaynak kodu.[18] Bu süreç, karmaşık yapılandırma adımları içerir ve en ufak bir sürüm veya yapılandırma uyuşmazlığında başarısız olmaya mahkumdur.

2.3. Tavsiye: Stratejik Bir Değişim

Yukarıdaki kanıtlar ışığında, ext-zmq eklentisini Windows üzerinde PHP 8.4 için kurma girişiminden vazgeçilmesi şiddetle tavsiye edilir.

Sonuç: PECL'de desteklenen bir sürümün olmaması, Windows üzerindeki DLL bağımlılıklarının kırılganlığı ve kaynaktan derlemenin aşırı karmaşıklığı, bu yolu üretim odaklı bir proje için aşırı riskli ve pratik olmayan bir hale getirmektedir. Bu yolda harcanacak zaman ve çaba, elde edilecek sonuca kıyasla orantısızdır ve başarı olasılığı düşüktür. Doğru mühendislik kararı, hedef platformda iyi desteklenen ve daha az kırılgan olan alternatif bir teknoloji seçmektir.

Bölüm 3: Veritabanı-Kuyruk Olarak Geri Çekilme Planının Eleştirel Değerlendirmesi

Kullanıcının ext-zmq kurulum sorununa karşı önerdiği alternatif çözüm, mevcut MySQL veritabanını bir mesajlaşma kuyruğu olarak kullanmaktır. Bu yaklaşım, yeni bir bağımlılık eklemeden mevcut altyapıyı kullanmanın getirdiği görünürdeki basitlik nedeniyle cazip gelebilir.[9] Ancak, bu "basitlik" yanılsaması, ciddi performans, ölçeklenebilirlik ve karmaşıklık sorunlarını gizlemektedir.

3.1. Veritabanı Kuyruğunun Çekici Basitliği

Bu modelin temel uygulaması oldukça basittir. Yayıncı süreç (HTTP işleyicisi), bir mesajı veritabanındaki bir tabloya INSERT eder. Tüketici süreç (WebSocket sunucusu) ise periyodik olarak bu tabloyu sorgulayarak (SELECT... WHERE status = 'new') yeni mesaj olup olmadığını kontrol eder ve işler.[21] İlk bakışta bu, ek bir sunucu veya kütüphane gerektirmediği için mantıklı bir çözüm gibi görünebilir.

3.2. Performans ve Ölçeklenebilirlik Karşıtı Model

Bu yaklaşımın "sağlam ve performanslı" bir sistem hedefiyle neden çeliştiğini anlamak için daha derine inmek gerekir.

- **Sorgulama Verimsizliği (Polling Inefficiency):** En büyük sorun, tüketicinin sürekli olarak veritabanını sorgulama zorunluluğudur. Hiçbir yeni mesaj olmasa bile, WebSocket sunucusu veritabanına sürekli olarak SELECT sorguları göndererek kaynakları israf eder.[9] Bu, "çekme" (pull) tabanlı bir modeldir. Buna karşılık, özel mesajlaşma sistemleri, mesaj geldiğinde bunu bekleyen tüketicilere anında "iten" (push) bir model kullanır, bu da çok daha verimlidir.
- **Gecikme ve Çekişme (Latency and Contention):** Veritabanları, düşük gecikmeli mesajlaşma için değil, veriyi kalıcı olarak depolamak ve sorgulamak için optimize edilmiştir. İşlem yönetimi (transaction management), indeksleme ve loglama gibi mekanizmaların getirdiği ek yük, mesaj teslimatında önemli gecikmelere neden olur.[8] Daha da kötüsü, aynı kuyruk tablosu üzerinde sürekli olarak yapılan okuma (SELECT) ve yazma (UPDATE/DELETE) işlemleri, kaynak çekişmesine (resource contention) yol açar. Bu durum, sadece kuyruk işlemlerini değil, uygulamanın diğer

tüm veritabanı operasyonlarını da yavaşlatır.[22]

- **Kilitleme Kabusları (Locking Nightmares):** Bu modelde depolama motoru seçimi hayati önem taşır. Eğer eski MyISAM motoru kullanılırsa (ki kesinlikle kullanılmamalıdır), bir işçinin bir mesajı işlemek için satırı güncellemesi, tüm tabloyu kilitleyerek (table-level lock) sistemi herhangi bir eşzamanlı yük altında durma noktasına getirir.[2] Modern InnoDB motoru satır seviyesinde kilitleme (row-level locking) ile daha iyi bir performans sunsa da, birden fazla işçinin aynı anda aynı mesajı almasını önlemek (yarış durumu - race condition) için SELECT... FOR UPDATE gibi özel kilitleme mekanizmaları kullanmak gerekir. Bu da karmaşıklığı artırır ve doğru yönetilmezse kilitlenmelere (deadlocks) yol açabilir.[21]
- **Bakım Karmaşıklığı:** Bu yaklaşım, özel mesaj kuyruklarında standart olarak gelen birçok özelliği manuel olarak geliştirmenizi gerektirir: işlenmiş görevlerin silinmesi (bu da zamanla tablo şişmesine ve parçalanmasına yol açar), başarısız olan görevlerin yönetimi, yeniden deneme mekanizmaları ve kalıcı olarak başarısız olan görevler için bir "ölü mektup kuyruğu" (dead-letter queue) oluşturma gibi karmaşık işlevler.[9]

3.3. Tablo: Özel Mesaj Kuyruğu ve Veritabanı Kuyruğu Karşılaştırması

Aşağıdaki tablo, iki yaklaşım arasındaki temel farkları özetleyerek veritabanı kuyruğunun neden bir anti-pattern olduğunu göstermektedir.

Özellik	Özel Mesaj Kuyruğu (Redis/RabbitMQ)	Veritabanı Olarak Kuyruk (MySQL)
Mesaj Teslimatı	**Push (İtme):** Mesajlar anında tüketicilere itilir.	**Poll (Çekme):** Tüketiciler sürekli veritabanını sorgular.
Gecikme	Çok düşük, milisaniye altı seviyelerde.	Yüksek, veritabanı işlem yükü nedeniyle.
Verim (Throughput)	Yüksek, saniyede on binlerce/yüz binlerce mesaj.	Düşük, veritabanı çekişmesi ve kilitleme ile sınırlı.
Eşzamanlılık	Dahili olarak yönetilir, kilitleme karmaşası yoktur.	Karmaşık, SELECT FOR UPDATE gibi manuel kilitleme gerektirir.

Ölçeklenebilirlik	Yatay olarak kolayca ölçeklenir.	Veritabanı darboğazı nedeniyle ölçeklenmesi zordur.
Güvenilirlik	Dahili yeniden deneme, ölü mektup kuyruğu özellikleri.	Tüm bu özelliklerin manuel olarak geliştirilmesi gerekir.
Bakım Yükü	Düşük, sistem bu iş için tasarlanmıştır.	Yüksek, özel mantık ve temizlik betikleri gerektirir.

3.4. Tavsiye: Yüksek Performanslı Sistemler İçin Kaçının

Sonuç olarak, bir veritabanı kuyruğu, çok düşük hacimli, kritik olmayan ve gecikmeye duyarlı olmayan arka plan görevleri için kabul edilebilir bir geçici çözüm olabilir.[20] Ancak, kullanıcının talep ettiği gibi "sağlam ve performanslı" gerçek zamanlı bir sistem için kesinlikle uygun değildir. Bu yola başvurmak, projenin ilerleyen aşamalarında kaçınılmaz olarak performans darboğazlarına, artan karmaşıklığa ve bakım kabuslarına yol açacaktır.

Bölüm 4: Önerilen Mimari: Sorunsuz IPC için Redis Pub/Sub'dan Yararlanma

ext-zmq'nun pratik olmaması ve veritabanı kuyruğunun bir anti-pattern olması nedeniyle, en sağlam ve performanslı çözüm, bu görev için özel olarak tasarlanmış bir aracı kullanmaktır. Bu bağlamda, Redis ve onun Publish/Subscribe (Pub/Sub) özelliği, kullanıcının ihtiyaçları için ideal bir denge sunar.

4.1. Publish-Subscribe Modelinin Tanıtımı

Redis Pub/Sub, süreçlerin birbirinden tamamen habersiz bir şekilde iletişim kurmasını

sağlayan bir mesajlaşma modelidir.[6] Bu model üç ana bileşenden oluşur:

- **Yayıncılar (Publishers):** Mesajları, kimin dinlediğini bilmeden, isimlendirilmiş "kanallara" (channels) gönderirler.[25]
- **Aboneler (Subscribers):** Bir veya daha fazla kanala ilgi duyduklarını belirtirler ve yayıncıları bilmeden bu kanallara gönderilen mesajları alırlar.[26]
- **Aracı (Broker - Redis):** Yayıncı ile abone arasındaki bu iletişimi yönetir, mesajları doğru kanallara yönlendirir ve abonelere dağıtır.

Bu model, kullanıcının mimarisine mükemmel bir şekilde uyar: Kısa ömürlü PHP-FPM süreci **yayıncı** olurken, uzun ömürlü Ratchet WebSocket sunucusu **abone** olur. Bu, aynı zamanda kullanıcının kendi araştırma dokümanında ölçeklendirme için önerilen modelle de birebir örtüşmektedir.[2]

4.2. Karşılaştırmalı Analiz: Redis Pub/Sub vs. ZeroMQ

Kullanıcının ilk tercihi olan ZeroMQ'dan neden Redis'e geçilmesi gerektiğini gerekçelendirmek kritik öneme sahiptir.

- **Mimari Model:** ZeroMQ, süreçler arasında doğrudan bağlantılar kuran aracısız bir kütüphanedir. Redis ise merkezi bir aracıdır.[28] Bu kullanım durumu için aracı modeli daha basittir: Hem PHP-FPM süreci hem de Ratchet süreci, birbirlerinin adresini veya varlığını bilmek zorunda kalmadan sadece merkezi Redis sunucusunun adresini bilmelidir. Bu, sistemin bileşenlerini birbirinden ayırır (decoupling) ve yönetimi kolaylaştırır.
- **Performans:** Kıyaslamalarda ZeroMQ'nun ham veriminin (~480k mesaj/saniye) Redis Pub/Sub'dan (~60-80k mesaj/saniye) daha yüksek olduğu doğru olsa da [28], bu fark bu uygulama için pratik bir anlam taşımaz. Redis'in performansı, bir QR giriş bildirim sistemi için fazlasıyla yeterlidir. Sistemin darboğazı büyük olasılıkla IPC mekanizması değil, uygulama mantığı veya veritabanı sorguları olacaktır.[30]
- **Platform Desteği ve Basitlik:** Bu, belirleyici faktördür. Redis, her platformda yaygın olarak kullanılan bir teknolojidir ve Windows üzerinde kolayca kurulabilen, kararlı PHP istemcilerine (phpredis eklentisi veya predis kütüphanesi) sahiptir.[31] Bu durum,
  ext-zmq'nun kurulum zorluklarıyla tam bir tezat oluşturur.
- **Ekosistem ve Çok Yönlülük:** Redis sadece bir Pub/Sub aracı değildir. Aynı zamanda yüksek performanslı bir bellek içi veri deposudur. Önbellekleme (caching), oturum yönetimi (session handling), oran sınırlama (rate limiting) gibi

görevler için de kullanılabilir. Bu, onu uygulama yığınına daha değerli ve çok yönlü bir katkı haline getirir.

4.3. Tablo: IPC Teknolojisi Karar Matrisi: ZeroMQ vs. Redis Pub/Sub

Aşağıdaki matris, ZMQ'dan Redis'e geçiş kararını gerekçelendirmek için temel kriterleri karşılaştırmaktadır.

Kriter	ZeroMQ	Redis Pub/Sub	Gerekçe ve Tavsiye
Mimari Model	Aracısız (Doğrudan Bağlantı)	Merkezi Aracı (Broker)	**Redis.** Merkezi model, bileşenleri ayırarak mimariyi basitleştirir.
Performans	Çok Yüksek	Yüksek	**Redis.** ZMQ daha hızlı olsa da, Redis'in hızı bu kullanım durumu için fazlasıyla yeterlidir.
Uygulama Karmaşıklığı	Orta (Soket yönetimi)	Düşük (Basit PUBLISH/SUBSCRIBE komutları)	**Redis.** API'si daha basittir ve daha az kod gerektirir.
Platform Desteği (PHP/Windows)	Çok Zayıf ve Kırılgan	Mükemmel ve Kararlı	**Redis.** Bu, en kritik faktördür. Redis, Windows'ta sorunsuz çalışır.
Güvenilirlik	En fazla bir kez (At-most-once)	En fazla bir kez (At-most-once)	**Eşit.** Her ikisi de temel pub/sub için benzer garanti sunar.
Ekosistem	Sadece Mesajlaşma	Mesajlaşma, Önbellek, Oturum Yönetimi vb.	**Redis.** Çok amaçlı olması, onu daha stratejik bir teknoloji seçimi yapar.

4.4. Alternatif Aracılar: RabbitMQ ve Mercure Üzerine Bir Not

- **RabbitMQ:** Daha karmaşık yönlendirme senaryoları ve garantili mesaj teslimatı gibi özellikler sunan son derece sağlam bir mesaj aracısıdır.[33] Ancak bu özellikler, anlık bir bildirim sistemi için genellikle gereğinden fazladır (overkill) ve daha karmaşık bir kurulum gerektirir.[35]
- **Mercure:** Server-Sent Events (SSE) tabanlı, PHP dostu modern bir alternatiftir.[36] Ancak, genellikle Caddy veya FrankenPHP gibi özel bir "hub" sunucusu gerektirir ve kullanıcının seçtiği WebSocket/Ratchet yolundan farklı bir mimari yaklaşım temsil eder.[38]

Bu alternatiflerin varlığını bilmek önemli olsa da, mevcut senaryo için Redis Pub/Sub, performans, basitlik ve platform kararlılığı arasında en iyi dengeyi sunan pragmatik çözümdür.

Bölüm 5: Redis Tabanlı Mimari için Adım Adım Uygulama Kılavuzu

Bu bölüm, önerilen Redis tabanlı mimariyi hayata geçirmek için pratik, adım adım talimatlar ve kod örnekleri sunmaktadır.

5.1. Ortam Yapılandırması: Redis'in Windows'ta Kurulumu ve Güvenliği

Redis'i üretim ortamında kullanmadan önce doğru şekilde kurulması ve güvenliğinin sağlanması kritik öneme sahiptir.

1. **Kurulum:** Windows için Redis'i kurmanın en güvenilir yolları, WSL2 (Windows Subsystem for Linux) üzerinde bir Linux dağıtımı kurup Redis'i oraya yüklemek veya Memurai gibi Windows için yerel ve kararlı bir Redis derlemesi kullanmaktır.
2. **Güvenlik Yapılandırması:** Kurulumdan sonra, redis.conf dosyasında aşağıdaki güvenlik adımları mutlaka uygulanmalıdır:
    - **Parola Koruması:** Redis'e erişimi parola ile korumak için requirepass

yönergesini etkinleştirin ve güçlü bir parola belirleyin. Bu, yetkisiz erişimi engelleyen ilk savunma hattıdır.[31]
`requirepass your_very_strong_password`

- **Ağ Arayüzü Kısıtlaması:** Redis'in yalnızca yerel makineden (localhost) gelen bağlantıları kabul etmesini sağlamak için bind yönergesini kullanın. Bu, Redis sunucusunun internete veya yerel ağdaki diğer makinelere yanlışlıkla maruz kalmasını önler.[39]
  `bind 127.0.0.1`

- **Tehlikeli Komutları Devre Dışı Bırakma:** Üretim ortamında, FLUSHALL, KEYS, CONFIG gibi tehlikeli komutları, boş bir dizeye yeniden adlandırarak devre dışı bırakmak iyi bir pratiktir.[41]
  `rename-command FLUSHALL ""`
  `rename-command KEYS ""`

5.2. Yayıncı: HTTP Sürecinden (PHP-FPM) Olayları Tetikleme

Mobil uygulamanın QR kodunu onayladığı /login/scan/confirm uç noktasını işleyen PHP denetleyicisi, Redis'e bir mesaj yayınlayacaktır.

1. **Redis İstemcisini Kurun:** Projenize Composer aracılığıyla bir Redis istemcisi ekleyin. Performans için phpredis PHP eklentisi şiddetle tavsiye edilir. Eğer eklenti kurma imkanınız yoksa, saf PHP ile yazılmış predis/predis kütüphanesi iyi bir alternatiftir.[32]
   Bash
   `pecl install redis # Veya`
   `composer require predis/predis`

2. **Yayınlama Kodu:** Aşağıdaki kod örneği, denetleyici içinde nasıl mesaj yayınlanacağını göstermektedir.
   PHP
   ```
   <?php
   // Gerekli sınıfları ve veritabanı bağlantısını dahil edin

   class AuthController
   {
   ```

```php
    public function confirmQrScan()
    {
        // Gelen isteği doğrulama mantığı...
        // $sessionId ve $userId'nin doğrulandığını varsayalım.
        $isValid = true; // Örnek
        $sessionId = $_POST['sessionId'];
        $userId = 123; // Örnek

        if ($isValid) {
            // Veritabanındaki oturum durumunu 'confirmed' olarak güncelle
            //...

            try {
                // Redis'e bağlan
                $redis = new Redis();
                $redis->connect('127.0.0.1', 6379);
                $redis->auth('your_very_strong_password');

                // Yayınlanacak mesajı hazırla
                $payload = json_encode();

                // 'qr-login-events' kanalına mesajı yayınla
                $redis->publish('qr-login-events', $payload);

                // HTTP yanıtını döndür
                http_response_code(200);
                echo json_encode(['status' => 'success']);

            } catch (Exception $e) {
                // Hata yönetimi
                http_response_code(500);
                echo json_encode(['status' => 'error', 'message' => 'IPC failed']);
            }
        } else {
            // Başarısız doğrulama yanıtı
        }
    }
}
```

?>
```

### 5.3. Abone: Gerçek Zamanlı Ratchet WebSocket Sunucusu

Bu, mimarinin en kritik parçasıdır. Ratchet sunucusu, hem WebSocket bağlantılarını yönetecek hem de Redis kanalını dinleyecektir.

1. **Gerekli Kütüphaneleri Kurun:** Ratchet ve asenkron bir Redis istemcisine ihtiyacınız olacak. clue/reactphp-redis, Ratchet'in temelindeki ReactPHP olay döngüsüyle sorunsuz entegre olduğu için mükemmel bir seçimdir.[42]
   Bash
   composer require cboden/ratchet clue/reactphp-redis

2. **Ratchet Sunucu Kodu:** Aşağıdaki tam ve açıklamalı kod, sunucunun nasıl oluşturulacağını gösterir.
   PHP
   ```php
   <?php
   require 'vendor/autoload.php';

   use Ratchet\MessageComponentInterface;
   use Ratchet\ConnectionInterface;
   use Ratchet\Server\IoServer;
   use Ratchet\Http\HttpServer;
   use Ratchet\WebSocket\WsServer;
   use React\EventLoop\Loop;

   class QrLoginServer implements MessageComponentInterface
   {
       protected $clients;
       protected $clientSessionMap;

       public function __construct()
       {
           $this->clients = new \SplObjectStorage;
           $this->clientSessionMap =;
   ```
   ```

```
        echo "WebSocket sunucusu başlatıldı.\n";
        $this->initializeRedisSubscriber();
    }

    protected function initializeRedisSubscriber()
    {
        // Asenkron Redis istemcisini oluştur
        $factory = new \Clue\React\Redis\RedisClientFactory();
        $redisClient =
$factory->createClient('redis://:your_very_strong_password@127.0.0.1:6379');

        $redisClient->on('error', function (Exception $e) {
            echo "Redis bağlantı hatası: ". $e->getMessage(). "\n";
            // Burada yeniden bağlanma mantığı eklenebilir (örn. exponential backoff)
        });

        $redisClient->on('close', function () {
            echo "Redis bağlantısı kapandı. Yeniden bağlanmaya çalışılacak...\n";
            // Basit bir yeniden bağlanma denemesi
            Loop::addTimer(5, function() {
                $this->initializeRedisSubscriber();
            });
        });

        $redisClient->subscribe('qr-login-events')->then(function () {
            echo "Redis 'qr-login-events' kanalına abone olundu.\n";
        });

        // 'message' olayı için dinleyici ekle
        $redisClient->on('message', function ($channel, $payload) {
            echo "Redis'ten mesaj alındı ($channel): $payload\n";

            $data = json_decode($payload, true);
            if (isset($data['sessionId']) && isset($this->clientSessionMap[$data['sessionId']]))
{
                $connection = $this->clientSessionMap[$data['sessionId']];
                echo "Hedef istemci bulundu. Mesaj gönderiliyor: {$data['sessionId']}\n";
                $connection->send($payload); // Gelen payload'ı doğrudan istemciye gönder
            }
```

```
        });
    }

    public function onOpen(ConnectionInterface $conn)
    {
        $this->clients->attach($conn);

        // Gelen bağlantıdan sessionId'yi al
        $queryParams =;
        parse_str($conn->httpRequest->getUri()->getQuery(), $queryParams);
        $sessionId = $queryParams['sessionId']?? null;

        if ($sessionId) {
            // sessionId'yi bağlantı nesnesiyle eşleştir
            $this->clientSessionMap[$sessionId] = $conn;
            echo "Yeni bağlantı ({$conn->resourceId}) sessionId: {$sessionId} ile açıldı.\n";
        } else {
            echo "Uyar: sessionId olmadan bir bağlantı açıldı. Kapatılıyor.\n";
            $conn->close();
        }
    }

    public function onMessage(ConnectionInterface $from, $msg)
    {
        // Bu senaryoda istemciden sunucuya mesaj beklenmiyor.
        // İsteğe bağlı olarak ping/pong için kullanılabilir.
    }

    public function onClose(ConnectionInterface $conn)
    {
        // Bağlantı kapandığında haritalardan kaldır
        foreach ($this->clientSessionMap as $sessionId => $connection) {
            if ($connection === $conn) {
                unset($this->clientSessionMap[$sessionId]);
                echo "Bağlantı kapatıldı: {$sessionId}\n";
                break;
            }
        }
        $this->clients->detach($conn);
```

```
    }

    public function onError(ConnectionInterface $conn, \Exception $e)
    {
        echo "Bir hata oluştu: {$e->getMessage()}\n";
        $conn->close();
    }
}

// Sunucuyu başlat
$server = IoServer::factory(
    new HttpServer(
        new WsServer(
            new QrLoginServer()
        )
    ),
    8080 // WebSocket sunucusunun çalışacağı port
);

$server->run();

?>
```

Bu yapı, HTTP ve WebSocket süreçleri arasında sağlam, ölçeklenebilir ve performanslı bir iletişim kanalı kurarak kullanıcının temel mimari sorununu etkin bir şekilde çözer.

Bölüm 6: Üretim Dağıtımı, Ölçeklendirme ve Operasyonlar

Geliştirilen mimariyi üretim ortamına taşımak ve gelecekteki büyümeyi desteklemek için ek operasyonel adımlar gereklidir. Bu bölüm, Ratchet sunucusunun sürekli çalışmasını sağlamak, ağ trafiğini doğru yönlendirmek ve sistemi yatay olarak ölçeklendirmek için en iyi uygulamaları detaylandırmaktadır.

6.1. Supervisor ile Süreç Yönetimi

Ratchet WebSocket sunucusu, bir komut satırı (CLI) uygulamasıdır. Bu, SSH oturumu kapatıldığında veya betik herhangi bir nedenle çöktüğünde sonlanacağı anlamına gelir. Üretim ortamında bu kabul edilemez. Bu sorunu çözmek için **Supervisor** gibi bir süreç kontrol sistemi kullanmak zorunludur.[2] Supervisor, Ratchet sunucusunu bir "daemon" (arka plan hizmeti) olarak çalıştırır, sürekli olarak izler ve bir hata durumunda otomatik olarak yeniden başlatır.[3]

Aşağıda, Ratchet sunucusu için üretim ortamına uygun, açıklamalı bir Supervisor yapılandırma dosyası örneği verilmiştir. Bu dosya genellikle /etc/supervisor/conf.d/ratchet-qr-server.conf gibi bir yola kaydedilir.

Ini, TOML

```ini
[program:ratchet-qr-server]
; Çalıştırılacak komut. PHP betiğinin tam yolu belirtilmelidir.
command=php /var/www/projeniz/bin/ratchet_server.php

; Süreç adı, loglarda ve komutlarda kullanılır.
process_name=%(program_name)s_%(process_num)02d

; Bu programdan kaç tane çalıştırılacağı. Tek sunucu için 1.
numprocs=1

; Supervisor başladığında programı otomatik olarak başlat.
autostart=true

; Program beklenmedik bir şekilde sonlanırsa otomatik olarak yeniden başlat.
autorestart=true

; Sürecin hangi UNIX kullanıcısı altında çalışacağı. Güvenlik için root olmamalıdır.
user=www-data

; Standart çıktı (stdout) loglarının yazılacağı dosya.
stdout_logfile=/var/log/supervisor/ratchet-qr.log

; Log dosyasının maksimum boyutu (örn. 50MB). Dolduğunda rotasyona uğrar.
stdout_logfile_maxbytes=50MB
```

```
; Kaç tane eski log dosyasının saklanacağı.
stdout_logfile_backups=10

; Standart hata (stderr) loglarının yazılacağı dosya.
stderr_logfile=/var/log/supervisor/ratchet-qr.err.log
stderr_logfile_maxbytes=50MB
stderr_logfile_backups=10
```

Bu yapılandırma, sadece süreci çalıştırmakla kalmaz, aynı zamanda log yönetimi gibi kritik üretim operasyonlarını da otomatikleştirir.[44]

6.2. Ağ Yapılandırması: Nginx ile Ters Proxy (Reverse Proxy)

Nginx, tüm gelen trafik için tek giriş noktası olarak görev yapmalı ve istekleri ilgili arka uç hizmetine yönlendirmelidir.[45] Bu, hem güvenlik hem de esneklik sağlar.

Aşağıda, hem standart web isteklerini PHP-FPM'e hem de WebSocket bağlantılarını Ratchet sunucusuna yönlendiren tam bir Nginx sunucu bloğu yapılandırması bulunmaktadır.

Nginx

```
# WebSocket bağlantılarını Ratchet sunucusuna yönlendirmek için bir upstream bloğu
upstream websocket_server {
    server 127.0.0.1:8080;
}

server {
    listen 443 ssl http2;
    server_name alanadiniz.com;

    # --- SSL/TLS Yapılandırması ---
    # Güvenli bağlantılar (HTTPS ve WSS) için zorunludur.
    ssl_certificate /path/to/your/fullchain.pem;
```

```
    ssl_certificate_key /path/to/your/privkey.pem;
    # Diğer SSL güvenlik ayarları...

    root /var/www/projeniz/public;
    index index.php index.html;

    # --- WebSocket İstekleri için Yönlendirme ---
    # /ws/ ile başlayan tüm istekler WebSocket olarak kabul edilir.
    location /ws/ {
        proxy_pass http://websocket_server;
        proxy_http_version 1.1;

        # WebSocket el sıkışması için gerekli olan başlıklar
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Bağlantının uzun süre açık kalması için okuma zaman aşımını artır
        proxy_read_timeout 86400;
    }

    # --- Standart HTTP İstekleri için Yönlendirme ---
    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    # PHP betiklerini PHP-FPM'e gönder
    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/var/run/php/php8.4-fpm.sock;
    }
}
```

Bu yapılandırma, Nginx'in SSL/WSS sonlandırmasını yapmasını, standart istekleri PHP-FPM'e, /ws/ ile başlayanları ise Ratchet sunucusuna sorunsuz bir şekilde

iletmesini sağlar.[46]

6.3. Mimariyi Ölçeklendirme: Tek Sunucunun Ötesi

Kullanıcı sayısı arttıkça, tek bir Ratchet sunucu süreci bir darboğaz haline gelebilir. Sistemi yatay olarak ölçeklendirmek, yani birden fazla Ratchet sunucusu çalıştırmak gerekir. Ancak WebSocket bağlantıları durum bilgisi içerdiği için bu basit bir görev değildir.[48]

- Çözüm 1 (Basit ama Kusurlu): Yapışkan Oturumlar (ip_hash)
  Nginx'in ip_hash yönergesi, aynı istemci IP adresinden gelen tüm isteklerin her zaman aynı arka uç sunucusuna gönderilmesini sağlar.49 Bu, WebSocket bağlantısının korunmasını sağlar. Ancak bu yöntemin ciddi kusurları vardır: Birçok kullanıcı aynı kurumsal ağ veya mobil operatör NAT'ı arkasındaysa, hepsi aynı sunucuya yönlendirilir ve bu da yükün dengesiz dağılmasına neden olur.39
- Çözüm 2 (Sağlam ve Önerilen): Paylaşılan Durum Mimarisi
  Bu, Bölüm 4'te önerilen Redis kullanımının stratejik bir uzantısıdır. Redis, sadece IPC için değil, aynı zamanda ölçeklendirme için de bir temel oluşturur. Bu mimari, kullanıcının kendi araştırma dokümanında da en gelişmiş çözüm olarak belirtilmiştir.2
  1. Birden fazla Ratchet sunucusu, Nginx arkasında standart bir yük dengeleme yöntemiyle (örneğin, round-robin) çalışır.
  2. Bir istemci herhangi bir Ratchet sunucusuna bağlandığında, o sunucu istemcinin sessionId'sini ve *kendi benzersiz sunucu kimliğini* Redis'te bir anahtar-değer olarak saklar.
  3. HTTP süreci bir giriş olayını yayınladığında, bunu belirli bir sunucuya değil, genel bir Redis Pub/Sub kanalına yayınlar.
  4. *Tüm* Ratchet sunucuları bu genel kanala abonedir.
  5. Bir Ratchet sunucusu mesajı aldığında, mesajdaki sessionId'nin Redis'te kendi sunucu kimliğiyle eşleşip eşleşmediğini kontrol eder.
  6. Yalnızca bağlantının sahibi olan sunucu, mesajı ilgili WebSocket istemcisine gönderir.

Bu model, WebSocket katmanını durumsuz hale getirir (durum Redis'te tutulur) ve gerçek, esnek ve hataya dayanıklı yatay ölçeklendirmeye olanak tanır.[51]

Tablo: Stateful WebSocket Bağlantıları için Nginx Yük Dengeleme Stratejileri

Strateji	Nginx Yapılandırması	Nasıl Çalışır?	Artıları	Eksileri	Tavsiye
Tek Sunucu	proxy_pass http://sunucu;	Tüm trafik tek bir sunucuya gider.	Çok basit kurulum.	Ölçeklenemez, tek hata noktası.	Sadece geliştirme ve çok küçük uygulamalar için.
Yapışkan Oturumlar	upstream { ip_hash;... }	Aynı IP'den gelen tüm bağlantıları aynı sunucuya yönlendirir.	Kolay yapılandırma.	Dengesiz yük dağılımı riski (NAT sorunu).	Orta ölçekli, NAT sorunu olmayan sistemler için geçici bir çözüm.
Paylaşılan Durum (Redis)	upstream { round_robin;... }	Yük dengeli dağıtılır. Sunucular durumu Redis üzerinden koordine eder.	Gerçek yük dengeleme, yüksek kullanılabilirlik, hataya dayanıklılık.	Daha karmaşık uygulama mantığı gerektirir.	**Üretim.** Sağlam ve ölçeklenebilir sistemler için en iyi uygulama.

Bölüm 7: Sonuç ve Nihai Mimari Tavsiyesi

7.1. Bulguların Sentezi

Bu rapor, kullanıcının ext-zmq eklentisini kurma sorunundan yola çıkarak, modern bir PHP uygulamasında süreçler arası iletişim (IPC) için en uygun mimariyi belirlemeyi

amaçlamıştır. Analiz süreci, birkaç temel bulguyu ortaya koymuştur:

1. **ext-zmq Yolu Pratik Değil:** Windows üzerinde PHP 8.4 gibi modern ve spesifik bir yapı için ext-zmq eklentisinin önceden derlenmiş kararlı bir sürümünü bulmak neredeyse imkansızdır. Kaynaktan derleme ise aşırı karmaşık ve risklidir. Bu nedenle bu yoldan stratejik olarak vazgeçmek, en doğru mühendislik kararıdır.
2. **Veritabanı Kuyruğu Bir Anti-Pattern'dir:** Mevcut bir veritabanını mesaj kuyruğu olarak kullanmak, ilk bakışta basit görünse de, sürekli sorgulama (polling), veritabanı üzerinde yarattığı aşırı yük, kilitleme sorunları ve bakım karmaşıklığı nedeniyle "sağlam ve performanslı" bir sistem hedefiyle tamamen çelişir.
3. **Redis Pub/Sub İdeal Çözümdür:** Redis Pub/Sub, performans, uygulama basitliği ve en önemlisi platform desteği ve kararlılığı arasında mükemmel bir denge sunar. PHP-FPM (yayıncı) ve Ratchet (abone) süreçleri arasındaki iletişimi sağlamak için hem pragmatik hem de teknik olarak sağlam bir çözümdür. Ayrıca, sadece bir IPC aracı olmakla kalmaz, aynı zamanda önbellekleme ve oturum yönetimi gibi diğer görevler için de kullanılarak uygulama yığınına değer katar.
4. **Mimari Seçimler Ölçeklenebilirliği Belirler:** Redis'in IPC için seçilmesi, sadece anlık sorunu çözmekle kalmaz, aynı zamanda gelecekteki yatay ölçeklendirme için de sağlam bir temel oluşturur. Paylaşılan durum mimarisi, Redis sayesinde mümkün hale gelir ve gerçek, hataya dayanıklı bir ölçeklendirme stratejisinin kapısını aralar.

7.2. Nihai Tavsiye

Yukarıdaki kapsamlı analiz ve değerlendirmeler ışığında, kullanıcının projesi için en sağlam, performanslı ve sürdürülebilir mimariyi oluşturmak üzere aşağıdaki adımların atılması kesin bir dille tavsiye edilir:

1. **Süreçler Arası İletişim (IPC):** PHP-FPM ve Ratchet süreçleri arasındaki iletişim için **Redis Pub/Sub** mekanizması kullanılmalıdır. Bu, react/zmq ve veritabanı kuyruğu alternatiflerine göre teknik olarak üstün ve pratik olarak uygulanabilir en iyi çözümdür.
2. **Dağıtım ve Yönlendirme:** Tüm gelen HTTP ve WebSocket trafiğini karşılamak, SSL/WSS sonlandırmasını yapmak ve istekleri doğru arka uç hizmetine (PHP-FPM veya Ratchet) yönlendirmek için **Nginx** bir ters proxy (reverse proxy) olarak yapılandırılmalıdır.
3. **Süreç Yönetimi:** Üretim ortamında Ratchet WebSocket sunucusunun sürekli çalışmasını, izlenmesini ve hata durumunda otomatik olarak yeniden başlatılmasını

sağlamak için **Supervisor** gibi bir süreç yöneticisi kullanılması zorunludur.

Bu mimari, yalnızca kullanıcının karşılaştığı mevcut teknik engeli etkin bir şekilde çözmekle kalmaz, aynı zamanda projenin gelecekteki büyüme ve ölçeklenme ihtiyaçlarına da yanıt veren, modern ve kanıtlanmış bir temel sağlar. Bu ilkelere bağlı kalmak, son kullanıcıya sorunsuz ve güvenli bir deneyim sunarken, geliştirme ve operasyon süreçlerini de basitleştiren sağlam bir sistemin ortaya çıkmasını temin edecektir.

Alıntılanan çalışmalar

1. How to Connect NGINX to PHP-FPM Using UNIX or TCP/IP Socket - Tecmint, erişim tarihi Temmuz 25, 2025, https://www.tecmint.com/connect-nginx-to-php-fpm/
2. PHP MySQL QR Giriş Sistemi_.pdf
3. Tutorial: Installation - Ratchet, erişim tarihi Temmuz 25, 2025, http://socketo.me/docs/deploy
4. zeromq/php-zmq - GitHub, erişim tarihi Temmuz 25, 2025, https://github.com/zeromq/php-zmq
5. ZeroMQ vs DBus for Pub-Sub pattern | Details - Hackaday.io, erişim tarihi Temmuz 25, 2025, https://hackaday.io/project/279-sonomkr-noise-monitoring/log/86364-zeromq-vs-dbus-for-pub-sub-pattern
6. Choosing the Right Messaging Tool: Redis Streams, Redis Pub/Sub, Kafka, and More, erişim tarihi Temmuz 25, 2025, https://dev.to/lovestaco/choosing-the-right-messaging-tool-redis-streams-redis-pubsub-kafka-and-more-577a
7. Scaling Pub/Sub with WebSockets and Redis - Ably, erişim tarihi Temmuz 25, 2025, https://ably.com/blog/scaling-pub-sub-with-websockets-and-redis
8. Using a Database as a Message Queue - GeeksforGeeks, erişim tarihi Temmuz 25, 2025, https://www.geeksforgeeks.org/system-design/using-a-database-as-a-message-queue/
9. Stop Using Your Database as a Message Queue: Why It's a Bad Idea - DEV Community, erişim tarihi Temmuz 25, 2025, https://dev.to/pratikpathak/stop-using-your-database-as-a-message-queue-why-its-a-bad-idea-2kff
10. Installing a PHP extension on Windows - Manual, erişim tarihi Temmuz 25, 2025, https://www.php.net/manual/en/install.pecl.windows.php
11. zmq - PECL :: Package - PHP, erişim tarihi Temmuz 25, 2025, https://pecl.php.net/package/zmq
12. Package :: zmq 1.1.3 for Windows - PECL - PHP, erişim tarihi Temmuz 25, 2025, https://pecl.php.net/package/zmq/1.1.3/windows
13. PHP For Windows: Home, erişim tarihi Temmuz 25, 2025, https://windows.php.net/

14. Adding the ZeroMQ PHP extension to XAMPP on Windows 10 and PHP7 - Stack Overflow, erişim tarihi Temmuz 25, 2025, https://stackoverflow.com/questions/40821137/adding-the-zeromq-php-extension-to-xampp-on-windows-10-and-php7
15. Installing zeromq on Windows 7 WAMP server - php - Super User, erişim tarihi Temmuz 25, 2025, https://superuser.com/questions/585291/installing-zeromq-on-windows-7-wamp-server
16. windows zmq dll issue for php 7.2 - Stack Overflow, erişim tarihi Temmuz 25, 2025, https://stackoverflow.com/questions/54012625/windows-zmq-dll-issue-for-php-7-2
17. ZMQ on Xampp (Windows) - php - Stack Overflow, erişim tarihi Temmuz 25, 2025, https://stackoverflow.com/questions/22854013/zmq-on-xampp-windows
18. Compiling ZMQ for PHP 8 NTS in Windows Server(IIS) - CMS Installation - Xibo Community, erişim tarihi Temmuz 25, 2025, https://community.xibo.org.uk/t/compiling-zmq-for-php-8-nts-in-windows-server-iis/32127
19. Building ZeroMQ on Windows - GitHub Gist, erişim tarihi Temmuz 25, 2025, https://gist.github.com/bseddon/c4938ce07d2e11cdecb744db0d06fc99
20. 'Don't use a database as a queue' : r/dotnet - Reddit, erişim tarihi Temmuz 25, 2025, https://www.reddit.com/r/dotnet/comments/1d8piwy/dont_use_a_database_as_a_queue/
21. What's the best way of implementing a messaging queue table in mysql - Stack Overflow, erişim tarihi Temmuz 25, 2025, https://stackoverflow.com/questions/423111/whats-the-best-way-of-implementing-a-messaging-queue-table-in-mysql
22. Why a database is not always the right tool for a queue based system - CloudAMQP, erişim tarihi Temmuz 25, 2025, https://www.cloudamqp.com/blog/why-is-a-database-not-the-right-tool-for-a-queue-based-system.html
23. Has anyone implemented a message queue with mysql and many workers? - Quora, erişim tarihi Temmuz 25, 2025, https://www.quora.com/Has-anyone-implemented-a-message-queue-with-mysql-and-many-workers
24. Performance considerations for using MySQL table as a queue? - Database Administrators Stack Exchange, erişim tarihi Temmuz 25, 2025, https://dba.stackexchange.com/questions/235402/performance-considerations-for-using-mysql-table-as-a-queue
25. Publish/subscribe system using redis;communicate between laravel and node services, erişim tarihi Temmuz 25, 2025, https://dev.to/niyiojeyinka/publish-subscribe-system-using-redis-communicate-between-laravel-and-node-services-152e
26. Redis Pub/sub | Docs, erişim tarihi Temmuz 25, 2025,

https://redis.io/docs/latest/develop/pubsub/
27. How I Tackled the WebSocket Scaling Challenge Economically | by Sidharth kumar samal, erişim tarihi Temmuz 25, 2025, https://medium.com/@sidharthkumarsamal/how-i-tackled-the-websocket-scaling-challenge-economically-015c6c5cd4e6
28. Comparison of ZeroMQ and Redis for a robot control platform - GitHub Gist, erişim tarihi Temmuz 25, 2025, https://gist.github.com/hmartiro/85b89858d2c12ae1a0f9
29. zmq vs redis for pub-sub pattern - Stack Overflow, erişim tarihi Temmuz 25, 2025, https://stackoverflow.com/questions/18591999/zmq-vs-redis-for-pub-sub-pattern
30. ZeroMQ vs node-ipc - Reddit, erişim tarihi Temmuz 25, 2025, https://www.reddit.com/r/node/comments/4k6dut/zeromq_vs_nodeipc/
31. Step-by-Step Guide: Installing and Configuring Redis as a PHP Session Handler on Ubuntu with PHP 8.3 - Akbar Sahata's Blog, erişim tarihi Temmuz 25, 2025, https://blog.akbarsahata.id/articles/step-by-step-guide-installing-and-configuring-redis-as-a-php-session-handler-on-ubuntu-with-php-8-3.md
32. Redis - Laravel 12.x - The PHP Framework For Web Artisans, erişim tarihi Temmuz 25, 2025, https://laravel.com/docs/12.x/redis
33. RabbitMQ vs Redis OSS - Difference Between Pub/Sub Messaging Systems - AWS, erişim tarihi Temmuz 25, 2025, https://aws.amazon.com/compare/the-difference-between-rabbitmq-and-redis/
34. RabbitMQ and WebSockets: Dynamic User Interfaces - Alibaba Cloud, erişim tarihi Temmuz 25, 2025, https://www.alibabacloud.com/tech-news/a/rabbitmq/gu0eyre34e-rabbitmq-and-websockets-dynamic-user-interfaces
35. RabbitMQ vs Redis | Svix Resources, erişim tarihi Temmuz 25, 2025, https://www.svix.com/resources/faq/rabbitmq-vs-redis/
36. dykyi-roman/server-client-interaction - GitHub, erişim tarihi Temmuz 25, 2025, https://github.com/dykyi-roman/server-client-interaction
37. Mercure: A WebSocket alternative for server-sent events - Hacker News, erişim tarihi Temmuz 25, 2025, https://news.ycombinator.com/item?id=42571651
38. Real-time - FrankenPHP: the modern PHP app server, erişim tarihi Temmuz 25, 2025, https://frankenphp.dev/docs/mercure/
39. Redis security | Docs, erişim tarihi Temmuz 25, 2025, https://redis.io/docs/latest/operate/oss_and_stack/management/security/
40. Setting Up a Redis Server as a Session Handler for PHP on Ubuntu 20.04 - CloudSigma, erişim tarihi Temmuz 25, 2025, https://blog.cloudsigma.com/setting-up-a-redis-server-as-a-session-handler-for-php-on-ubuntu-20-04/
41. Securing Your Redis Kingdom: Essential Best Practices - Security Insights, erişim tarihi Temmuz 25, 2025, https://blog.securityinsights.io/redis-production-security-checklist
42. Introducing async Redis database client for ReactPHP - clue·engineering, erişim tarihi Temmuz 25, 2025, https://clue.engineering/2019/introducing-reactphp-redis

43. Supervisor Guide for PHP Developers - DEV Community, erişim tarihi Temmuz 25, 2025, https://dev.to/edgaras/supervisor-guide-for-laravel-developers-configuration-and-use-cases-20i4
44. Logging — Supervisor 4.2.5 documentation, erişim tarihi Temmuz 25, 2025, https://supervisord.org/logging.html
45. NGINX Reverse Proxy | NGINX Documentation, erişim tarihi Temmuz 25, 2025, https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/
46. WebSocket proxying - nginx, erişim tarihi Temmuz 25, 2025, http://nginx.org/en/docs/http/websocket.html
47. Using nginx to host websockets and http on the same domain:port | by Tom van Neerijnen | localhost.run | Medium, erişim tarihi Temmuz 25, 2025, https://medium.com/localhost-run/using-nginx-to-host-websockets-and-http-on-the-same-domain-port-d9beefbfa95d
48. How to Handle WebSocket Load Balancing Without Losing the Connection Thread, erişim tarihi Temmuz 25, 2025, https://blog.thnkandgrow.com/how-to-handle-websocket-load-balancing-without-losing-connection-thread/
49. Using nginx as HTTP load balancer, erişim tarihi Temmuz 25, 2025, http://nginx.org/en/docs/http/load_balancing.html
50. load-balancing.conf - nicokaiser/nginx-websocket-proxy - GitHub, erişim tarihi Temmuz 25, 2025, https://github.com/nicokaiser/nginx-websocket-proxy/blob/master/load-balancing.conf
51. Is `ip_hash` required to proxy websockets? : r/nginx - Reddit, erişim tarihi Temmuz 25, 2025, https://www.reddit.com/r/nginx/comments/7159qo/is_ip_hash_required_to_proxy_websockets/