MUHAMMET ŞAFAK — GÖRSELLER

Veri yoğunluklu sistemlerde 5 kırılma noktası

En ucuz müdahaleden en pahalısına bir sıra ve her bileşeni gerekçeli bir Laravel production stack'i.

Muhammet Şafak • 3 Ekim 2026

BÖLÜM 01

Beş kırılma noktası

Working set'ten ayrı veri katmanına: en ucuz müdahaleden en pahalısına.

ÖLÇEĞİ BELİRLEYEN

Request değil, veri profili.

Bir sistem dakikada 50 request alıp yine de veri tarafından boğulabilir; bir başkası saniyede 5.000 request'i rahat karşılarken veri katmanı el değmemiş durur.

VERİ PROFİLİ

Beş eksen

Working set / RAM oranı

Sık okunan veri belleğe sığıyor mu?

Okuma / yazma oranı

Yük hangi yöne ağır basıyor?

Tek tablo boyutu

Tek bir tablo ne kadar büyüdü?

Yazma throughput'u

Primary saniyede ne kadar yazıyor?

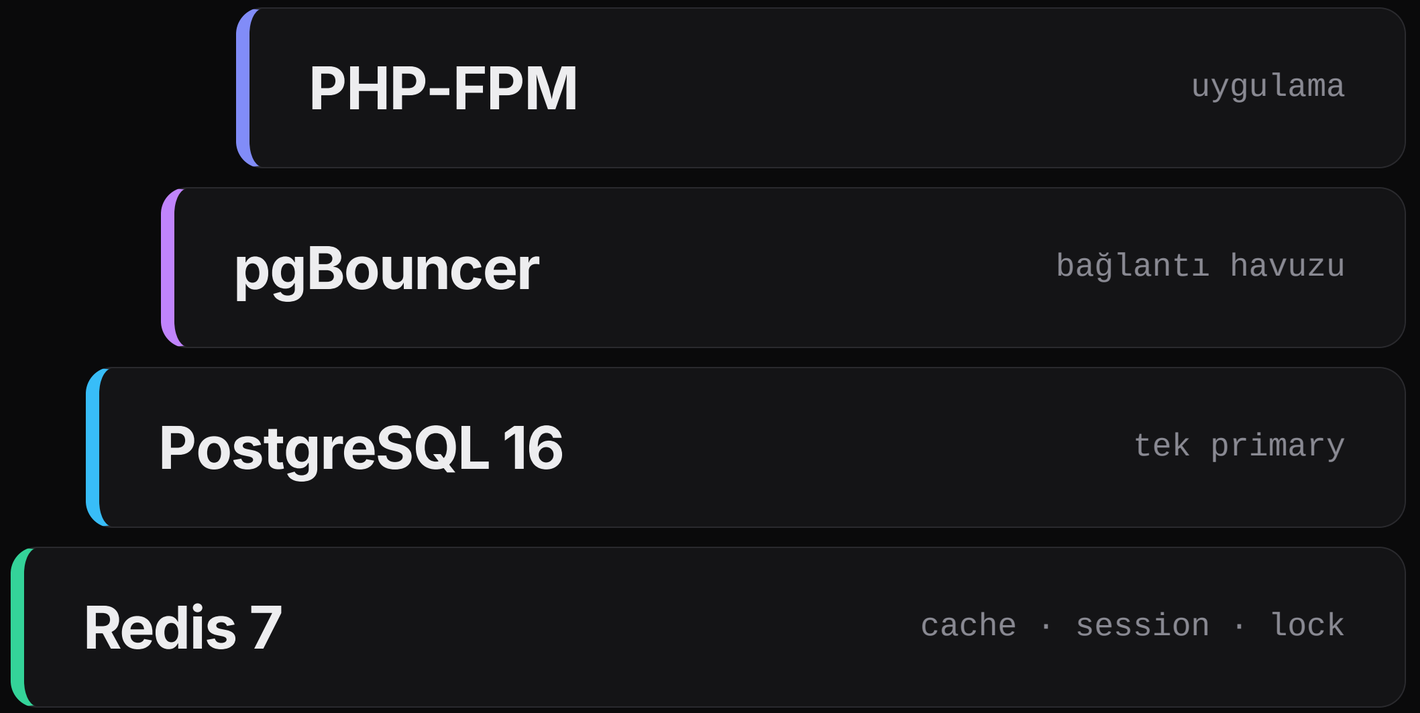
Tutarlılık gereksinimi

Hangi okuma en güncel veriyi görmek zorunda?

BAŞLANGIÇ MİMARİSİ

Tek primary ile başlamak

PHP-FPM, PostgreSQL'e pgBouncer üzerinden, Redis'e doğrudan bağlanır. Birkaç on GB veri ve dakikada binlerce request için yeterli; working set RAM'e sığdığı sürece.



KIRILMA NOKTASI 01 · WORKİNG SET

Sinyal: cache hit ratio

Sık okunan veri RAM'e sığmıyor. %99 üzeri sağlıklı. %95'e doğru düşüş, disk IOPS'u ile birlikte okunduğunda working set'in RAM'den taştığını gösterir.

pg_stat_database.sql

```sql
SELECT
sum(blks_hit) * 100
  / nullif(sum(blks_hit) + sum(blks_read), 0)
  AS cache_hit_ratio
FROM pg_stat_database;
```

MÜDAHALE SIRASI

Önce ucuz olanı tüketin

1

Index disiplini

`gereksiz taramayı kes`

2

Ölü veriyi ayır

`sıcak veriyi küçült`

3

Vertical scale

`en son, daha çok RAM`

KIRILMA NOKTASI 02 · OKUMA YÜKÜ

Okuma replica'ya, yazma primary'ye

Sinyal: CPU yüksek, SELECT baskın, pgBouncer'da `cl_waiting` birikiyor. `sticky`, aynı request içinde yazmadan sonraki okumaları primary'ye yollar.

config/database.php

```
'pgsql' => [
  'driver'  => 'pgsql',
  'read'  => ['host' => ['10.0.0.2']],   // replica
  'write' => ['host' => ['10.0.0.1']],   // primary
  'sticky' => true,
  // ...ortak ayarlar
],
```

BEDELİ

Replica'nın getirdiği yeni sorun

GECİKME Replication lag: replica geride kalır; read-after-write garantisi kaybolur.

İZLEYİN pg_stat_replication'da replay_lsn farkını lag_bytes olarak okuyun.

PRİMARY Bakiye, stok, yetki gibi tutarlılık isteyen okumalar primary'de kalır.

SINIR Read replica yazma kapasitesine hiçbir şey katmaz.

KIRILMA NOKTASI 03 · TEK TABLO

Zamana göre bölünen tablo

Sinyal: `n_dead_tup` birikiyor, `last_autovacuum` geride kalıyor. Partition'ları elle açmak yerine `pg_partman` kullanılabilir.

events.sql

```sql
CREATE TABLE events (
  id          bigserial,
  occurred_at timestamptz NOT NULL,
  payload     jsonb       NOT NULL
) PARTITION BY RANGE (occurred_at);

CREATE TABLE events_2026_05 PARTITION OF events
  FOR VALUES FROM ('2026-05-01') TO ('2026-06-01');
```

KAZANIM VE KOŞUL

Partitioning ne kazandırır?

Partition pruning

Sorgu yalnız ilgili partition'a bakar.

Ucuz silme

Eski veri DROP TABLE events_2026_01 ile gider.

Sınırlı vacuum

Vacuum tüm tabloyu değil, partition'ı tarar.

Primary key

Partition anahtarını içermeli: (id, occurred_at).

BRIN

Zamana göre sıralı veride küçük ve ucuz index.

Sinyal: I/O wait, checkpoint ve WAL baskısı

1

Amplifikasyonu azalt

`idx_scan = 0 index'leri sil, fillfactor/HOT, batch insert`

2

Append verisini çıkar

`append ağırlıklı veri primary'den ayrılır`

3

Veri alanını böl

`en son: functional sharding`

GERÇEKÇİ BEKLENTİ

Çoğu sistem 4. noktaya varmaz.

Acele etmeyin: çoğu sistem 4. noktaya hiç ulaşmaz, ulaşanların çoğu da aslında 1. noktayı (eksik index, şişmiş working set) atlamış olduğu için erken gelir.

KIRILMA NOKTASI 05

Ayrı veri katmanı

HAMLE Veri-baskın alan ayrı bir PostgreSQL instance'ına taşınır.

GEÇİŞ pgBackRest restore, logical replication ve kısa bir cutover.

BEDEL İkinci backup, monitoring ve upgrade; iki veri arasında JOIN yok.

KURAL Ayırma kararı veri profiliyle verilir, organizasyon şemasıyla değil.

YANLIŞ TEŞHİS

Yanlış kırılma noktaları

GÖRÜNEN	GERÇEK	DOĞRU HAMLE
"Veritabanı yavaş"	N+1: uygulama 200 sorgu atıyor	Eager loading; sorguları say
"Replica lazım"	Tek sorguda eksik index	EXPLAIN ANALYZE, index ekle
"Sharding lazım"	Replica, partitioning denenmedi	Önce 2. ve 3. noktayı tüket
"NoSQL'e geçelim"	jsonb, BRIN, partitioning atıl	PostgreSQL'i sonuna kadar kullan
"Yazma için cache"	Cache okumayı azaltır, yazmayı değil	4. noktaya bak

ORTAK HATA KALIBI

Hiçbiri yazma kapasitesi eklemez.

Read replica okumayı ölçekler. Cache okumayı azaltır. Partitioning tek tabloyu evcilleştirir. Hiçbiri yazma throughput'unu artırmaz.

JSONB · LISTEN/NOTIFY · PARTİTİONİNG · BRIN · LOGİCAL REPLİCATİON

%20'si kullanıldı. Bu bir sınır değil.

Çoğu ekip PostgreSQL'in %20'sini kullanıp "PostgreSQL yetmiyor" sonucuna varır.

ÖZET

En ucuzdan en pahalıya

1

Working set

`index + RAM`

2

Okuma yükü

`read replica`

3

Tek tablo

`partitioning`

4

Yazma yükü

`amplifikasyon + ayrıştırma`

5

Veri-baskın alan

`ayrı veri katmanı`

BÖLÜM 02

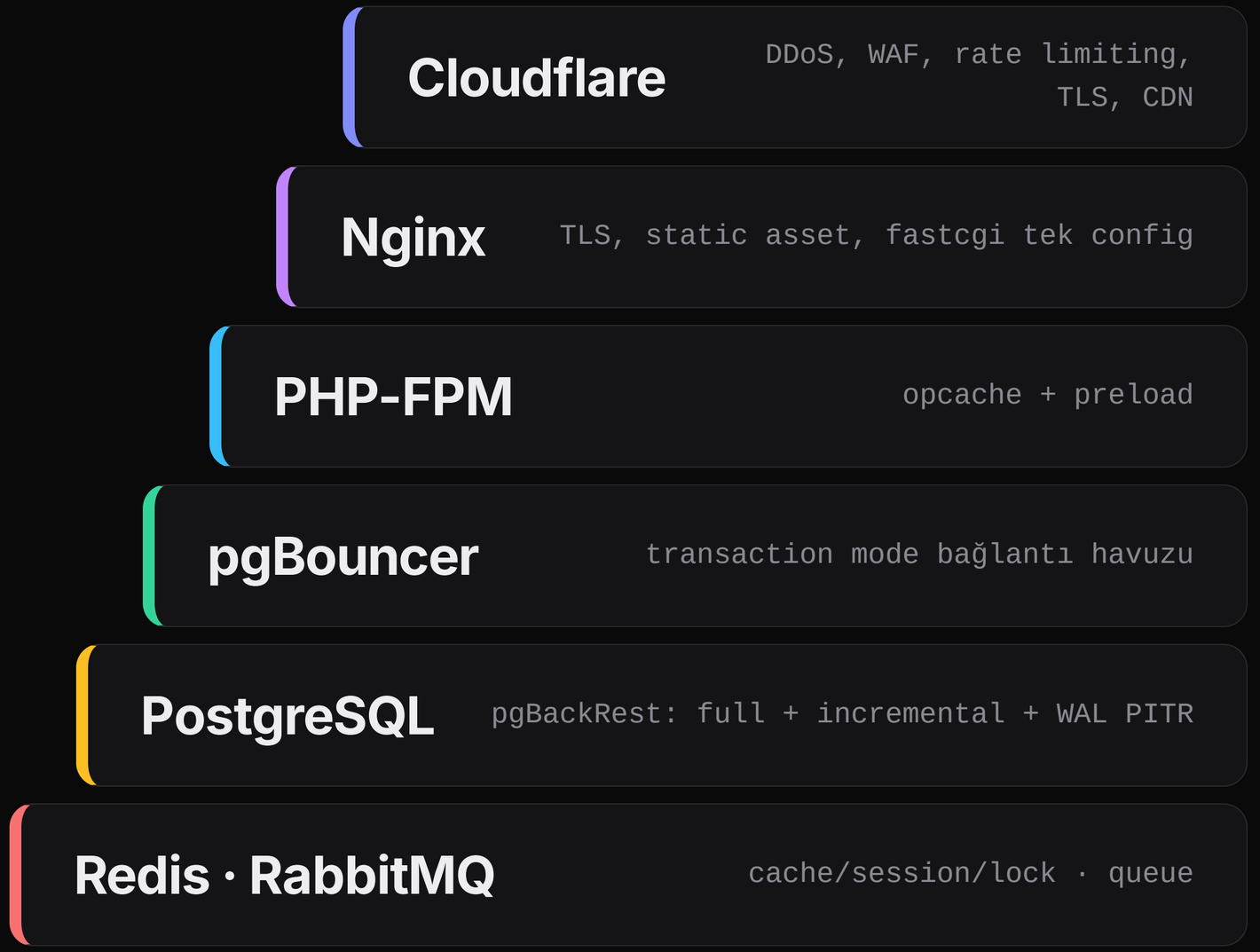
Laravel stack

"Her şey gerekli" demiyorum — gerekli olduğunu anlayana kadar minimum tutmayı savunan biriyim — ama her bileşenin kazandırdığı somut.

ANATOMİ

Her bileşen neden orada?

İstek Cloudflare'den PHP-FPM'e iner. PHP-FPM üç arka servise ayrı bağlanır: pgBouncer üzerinden PostgreSQL, Redis ve RabbitMQ. Worker'lar Supervisor altında Horizon ile koşar.



REDİS

Tek instance, üç şapka

Cache (`Cache::remember`), session (`SESSION_DRIVER=redis`) ve lock (`Cache::lock`) tek Redis'te güvenle çalışır; yeter ki bu üç ayar yerinde olsun.

redis.conf

```
maxmemory-policy volatile-lru
appendonly no
save 900 1 300 10 60 10000
```

QUEUE (OPSİYONEL)

RabbitMQ ile Redis kuyruğu

	Redis	RabbitMQ
PERSİSTENCE	Bellekten servis eder; diske RDB ya da AOF ile iner	✓ Kalıcı mesajları diske yazar
YENİDEN TESLİM	retry_after zaman aşımına bakar	✓ Consumer'ın bağlantısına bakar
ROUTİNG	Yönlendirme kararı uygulamada kalır	✓ Exchange'ler karar verir
MONİTORİNG	Horizon; uygulamanın dispatch ettiğiyle sınırlı	✓ Broker düzeyinde yönetim paneli

ÇIKARINCA OLAN

Hangi parçası kaldırılırsa ne kaybedersiniz?

BİLEŞEN	ÇIKARINCA OLAN
pgBouncer	Connection storm'larında PostgreSQL bağlantısı tükenir, request'ler 500 atar
Supervisor	Worker'lar crash sonrası geri gelmez, alarm kurmak zorundasınız
Horizon	Görünürlük çöker, "queue neden yavaş" sorusunu kör cevaplarsınız
opcache	Her request PHP dosyasını disk'ten okur ve parse eder, ~5x yavaşlama
pgBackRest	pg_dump kalır: PITR gider, incident'ta hata payınız bir günlük yedek
Redis lock	Race condition'lara açıksınız, distributed-safe mutex'iniz kalmaz

Nicel değerli, değişimi kolay.

Boring stack'in gücü tam burada: her parça nicel olarak değerli ve değişimi kolay.

Teşekkürler

sade.dev

Önce veri profilini ölçün, sonra en ucuz müdahaleyi seçin.