PHP İle Gelişmiş Cli Worker ve Event Yönetimi

Beytullah GÜRPINAR

Head of Engineering - Zotlo

PHPkonf.

TEKNASYON

zotlo.

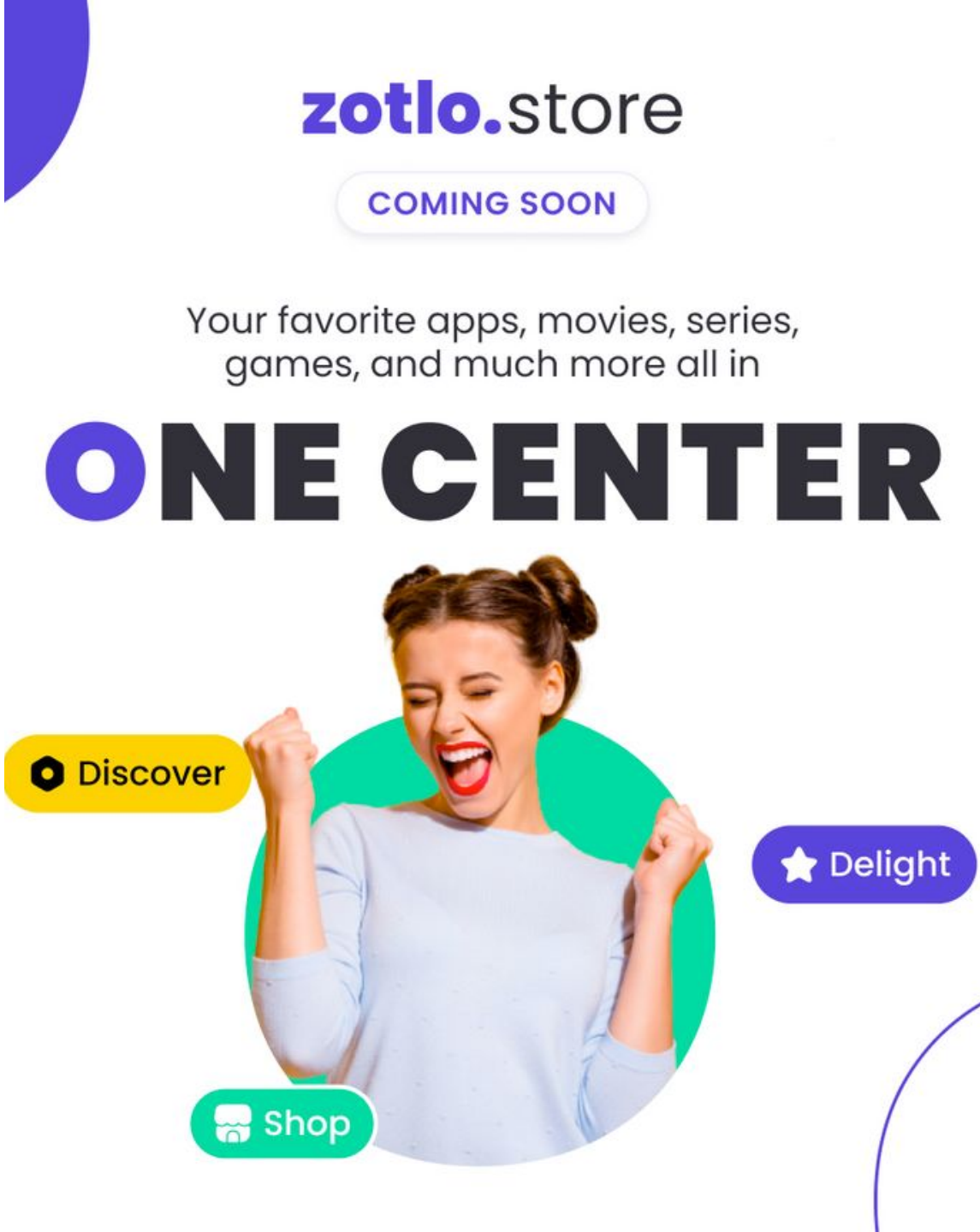
- Uçtan Uca “web-to-app” Deneyimi
- Global Ödeme Yöntemleri
- No Code Abonelik Platformu
- Abonelik Yönetimi

30M+ Abonelik Süreci

20K Dakikalık Ödeme Denemesi

30+ Ödeme Sağlayıcı Entegrasyonu

50M+ Aylık Landing Ziyareti



Worker Nedir?

- Worker'lar, arka planda çalışan ve belirli görevleri otomatikleştiren bağımsız servislerdir.
- Web siteleri, uygulamalar ve diğer sistemler tarafından çeşitli görevleri yerine getirmek için kullanılırlar.
- Tek bir worker olarak ya da paralel olarak birden fazla worker birlikte çalışabilir.
- Uzun süreli işlerde, veriyi hazırlayarak ön yüzün kullanabileceği hale getirmek için arka planda çalışırlar.
- Ana proje dilinden farklı bir dilde hazırlanabilirler. Bu şekilde ana servis üzerindeki dil bağımlılığını ortadan kaldırabilirler.

Worker Mimarileri

Single Worker

- Bir görevin yalnızca tek bir worker ile yapıldığı yapılar
- Daha çok basit arka plan işleri için tercih edilir.
- Bu yapılarda race condition riski düşük seviyededir.
- Yönetimi daha kolaydır ve bir hata durumunda kolaylıkla durdurulabilir.
- Hataları yakalamak ve izlemek kolaydır.

Concurrent Worker

Concurrent worker'lar, single worker'lara kıyasla, bir servisin aynı anda paralel olarak istenilen sayıda çalıştırılmasına olanak tanıyan servislerdir. Bu servisler, büyük veri kümelerini veya karmaşık işleri çok daha hızlı bir şekilde paralel olarak tamamlamak için kullanılır.

Concurrent Worker'ların Avantajları:

- **Hız:** Büyük veri kümelerini veya karmaşık işleri çok daha hızlı işleyebilirler.
- **Ölçeklenebilirlik:** Kullanılan ortamlarla hızlı bir şekilde iş yüküne göre worker sayısı arttırılabilir veya azaltılabilir.
- **Maliyet etkinliği:** İş yüküne göre kaynakları daha verimli kullanarak maliyetleri düşürebilirler.
- **Güvenilirlik**: Bir worker arızalanırsa, diğer worker'lar işleri işlemeye devam edebilir. Bu, sistemin daha güvenilir ve dayanıklı olmasını sağlar.
- **Kullanım Kolaylığı**: Concurrent worker'lar, genellikle bir kuyruk yönetim sistemi ile birlikte kullanılır. Bu, worker'ları yönetmeyi ve işleri işlemelerini sağlamayı kolaylaştırır.

Worker Mimarileri

Örnek Senaryo

- Bir servise ait aboneliklerin yönetilmesi.
- Single worker kullanarak abonelik ödemelerinin yönetilmesi
- Concurrent mimarisi ile abonelik ödemelerinin yönetilmesi
  - Worker içerisinde event yönetimi.
  - Worker üzerinden event kullanımı ile ödeme alınması.

TEKNASYON

Single Worker

```php
class SubscriptionRenewalTask extends Task
{
    private $jobId = null;
    private $workerName = 'ChargeTask';
    protected $shouldStop = false;

    public function mainAction()
    {
        declare(ticks=1);
        pcntl_signal(SIGTERM, [$this, 'stop']);
        pcntl_signal(SIGINT, [$this, 'stop']);

        while (true) {
            $this->jobRunningCheck();
            $subscriptions = Subscriber::find(
                [
                    'conditions' => 'renewal_date <= :renewalDate AND status = :statusActive',
                    'bind' => [
                        'renewalDate' => new \DateTime('now', new \DateTimeZone('UTC')),
                        'statusActive' => 'active'
                    ],
                    'limit' => 10
                ]
            );

            foreach ($subscriptions as $subscriber) {
                $this->jobRunningCheck();
                $this->jobId = Uuid::uuid4();

                $subscriber->setRenewalDate(new \DateTime('+1 day', new \DateTimeZone('UTC'))); // Set renewal date to next day
                $subscriber->save();

                if (!$this->lockTask($subscriber)) {
                    continue;
                }

                try {
                    $this->logger->info('Charging subscriber', ['subscriberId' => $subscriber->getId(), 'jobId' => $this->jobId, 'workerName' => $this->workerName]);
                    Payment::charge($subscriber);
                    $this->unlock($subscriber->getId());
                } catch (\PaymentError $e) {
                    $this->logger->error('Error charging subscriber', ['subscriberId' => $subscriber->getId(), 'jobId' => $this->jobId, 'error' => $e->getMessage()]);
                    $this->unlock($subscriber->getId());
                } catch (\Exception $e) {
                    $this->logger->error('Error charging subscriber', ['subscriberId' => $subscriber->getId(), 'jobId' => $this->jobId, 'error' => $e->getMessage()]);
                }
            }
        }

    }
}
```

TEKNASYON

Örnek PHP Concurrent Worker - Sender

```php
public function sendQueueAction()
    {
        declare(ticks=1);
        pcntl_signal(SIGTERM, [$this, 'stop']);
        pcntl_signal(SIGINT, [$this, 'stop']);

        while (true) {
            $this->jobRunningCheck(); // Check if the job is running
            $subscriptions = Subscriber::find(
                [
                    'conditions' => 'renewal_date <= :renewalDate AND status = :statusActive',
                    'bind' => [
                        'renewalDate' => new \DateTime('now', new \DateTimeZone('UTC')),
                        'statusActive' => 'active'
                    ],
                    'limit' => 10
                ]
            );

            foreach ($subscriptions as $subscription) {
                $this->jobId = Uuid::uuid4();
                $subscription->setRenewalDate(new \DateTime('+1 day', new \DateTimeZone('UTC'))); // Set renewal date to next day
                $subscription->save();
                $this->logger->info('Processing subscriber queue', ['subscriberId' => $subscription->getId(), 'jobId' => $this->jobId]);
                QueueManagement::sendChargeQueue($subscription, QueueManagement::getQueueName($subscription->getMerchantId()));
            }
        }
    }
```

TEKNASYON

Örnek PHP Concurrent Worker Process

```php
private function processQueueAction($queueName = 'default')
    {
        $queueManager = (new QueueManager())->getRenewalClient([
            'queueName' => $queueName,
            'maxNumberOfMessages' => 10, // 10 adet mesaj alacak
            'visibilityTimeout' => 60 // Silinmediği takdirde 60 saniye sonra tekrar görünür olacak
        ]);
        while (true) {
            $this->jobRunningCheck();
            $messages = $queueManager->receive();

            if (is_array($messages) && count($messages) > 0) {
                foreach ($messages as $message) {

                    $this->jobId = Uuid::uuid4();

                    $data = json_decode($message['Body'], true);
                    $this->logger->info('Processing subscriber', ['subscriberId' => $data['subscriberId'], 'jobId' => $this->jobId, 'workerName' => $this->workerName]);

                    if (!$this->lock($data['subscriberId'])) {
                        $this->logger->error('Subscriber is already processing', ['data' => $data, 'jobId' => $this->jobId, 'workerName' => $this->workerName]);
                        continue;
                    }

                    try {
                        Payment::chargeQueue($data);
                        QueueManager::deleteMessage($message);

                    } catch (\PaymentError $e) {
                        $this->unlockTask($data['subscriberId']);
                        $this->logger->error('Error charging subscriber', ['subscriberId' => $data, 'jobId' => $data['jobId'], 'error' => $e->getMessage()]);
                    } catch (\Exception $e) {
                        $this->logger->error('Error charging general error', ['data' => $data, 'jobId' => $data['jobId'], 'error' => $e->getMessage()]);
                    }
                }
            }
        }
    }
```

Race Condition - Stop Start

```php
pcntl_signal(SIGTERM, [$this, 'stop']);
pcntl_signal(SIGINT, [$this, 'stop']);

private function lockTask($subscriberId)
{
    $lockKey = 'lock:subscription:' . $subscriberId;
    if ($this->redis->setnx($lockKey, 1)) {
        $this->redis->expire($lockKey, 60);
        return true;
    }
    return false;
}

private function unlockTask($subscriberId)
{
    $lockKey = 'lock:subscription:' . $subscriberId;
    $this->redis->del($lockKey);
}

private function stop()
{
    $this->shouldStop = true;
}

private function jobRunningCheck()
{
    if ($this->shouldStop) {
        $this->logger->info('Worker Terminate: ' . $this->workerName . ', ' . gmdate('Y-m-d H:i:s'));
        exit('Worker Terminate: ' . $this->workerName . ', ' . gmdate('Y-m-d H:i:s') . PHP_EOL);
    }
}
```

TEKNASYON

Dikkat Edilmesi Gerekenler

Race Condition

Bir işi aynı anda birden fazla workerin alabilmesi durumudur. Dikkat edilmezse aynı iş duplicate şekilde yapılır.

Hata Yakalama

Worker içerisinde olabilecek tüm hata senaryoları düşünülmeli ve bu hatalar raporlanmalıdır.

Rollback Mimarisi

Bir hata durumunda ilgili işlemin durdurulması ya da geriye alınması ile ilgili senaryolar mutlaka düşünülmelidir.

Performans - Ölçeklenebilirlik

Workerları çoklu kullanmak demek sınırsız sayıda kullanabilmek demek değildir. Diğer kaynaklar da göz önüne alınarak ölçeklendirilmelidir.

Güvenlik

Paralelde çok fazla işlem yapılacağından küçük bir güvenlik açığı tüm workerları etkileyecektir.

Acil Durum Sistemi

Acil bir problemde workerların stop edilmesi senaryoları planlanmalıdır.

Maliyet

TEKNASYON

Kuyruk Yönetim Sistemleri

Bu sistemler, işleri sıraya koyarak ve işleri eş zamanlı olarak dağıtarak iş akışlarını iyileştirmeye ve karmaşıklığı azaltmaya yardımcı olur.

Kuyruk Yönetim Sistemlerinin Avantajları:

Ölçeklenebilirlik: Kuyruk yönetim sistemleri, iş yüküne göre ölçeklendirilebilir.

Yük Dengeleme: Kuyruk yönetim sistemleri, işleri birden fazla işçiye dağıtarak yükü dengeleyebilir.

Hata Yönetimi: Kuyruk yönetim sistemleri, hataları otomatik olarak yeniden deneme ve hataya neden olan işleri tekrar işleme koyma gibi hata yönetimi özellikleri sunabilir.

Monitoring: Kuyruk yönetim sistemleri, iş akışlarını izlemek ve işlerin durumunu takip etmek için araçlar sunabilir.

Popüler Kuyruk Yönetim Sistemleri:

AWS SQS, Apache Kafka, RabbitMQ, Google Pub/SUb

Özellik	Apache Kafka	Amazon SQS
Kullanım tipi	Event streaming platform	Mesaj kuyruğu hizmeti
Kullanım alanı	Büyük hacimli veri işleme ve gerçek zamanlı veri analizi.	Dağıtılmış sistemlerde basit kuyruk işleme.
Ölçeklendirme	Yüksek oranda ölçeklenebilir, veri depolama ve akış için tasarlanmıştır.	Yüksek oranda ölçeklenebilir, yüksek yükleri idare edebilir.
Kuyruk yapısı	Kafka konu (topic) olarak adlandırılan kuyruğu kullanır ve genellikle birden fazla aboneye aynı anda veri göndermeyi sağlar.	Geleneksel kuyruk yapısını takip eder ve bir alıcı için tasarlanmıştır.
Veri saklama	Verileri disk üzerinde saklayarak depolar ve düzenli bir şekilde veri akışı sağlar.	Mesajları kısa süreli saklar (4 gün ile 14 gün arasında).
Kullanım zorluğu	Daha karmaşık bir yapı, kurulumu ve yönetimi daha fazla çaba gerektirir.	Kullanımı ve kurulumu daha kolaydır, AWS yönetimiyle entegredir.
Gecikme	Düşük gecikme, gerçek zamanlı uygulamalar için uygundur.	Tipik olarak düşük gecikme, hızlı kuyruk işleme sağlar.
Fiyatlandırma	Donanım, altyapı ve yönetim maliyetleri gerektirir.	Kullanım başına ödeme modeliyle maliyetler kontrol edilebilir.
Yönetim	Donanım ve yazılım yönetimi kullanıcının sorumluluğundadır.	AWS tarafından yönetilir, kullanıcının yönetimi kolaylaştırılır.

TEKNASYON

Supervisord

Supervisord, birden fazla işlemi (programı) otomatik olarak başlatmak, izlemek ve yönetmek için kullanılan, python ile yazılmış ve açık kaynaklı bir projedir.

Supervisord'un Avantajları:

Otomasyon: İşlemleri otomatik olarak başlatabilir, durdurabilir ve yeniden başlatabilir.

İzleme: Çalışan işlemleri izleyebilir ve CPU, bellek ve ağ kullanımı gibi kaynak kullanımı hakkında bilgi sağlayabilir.

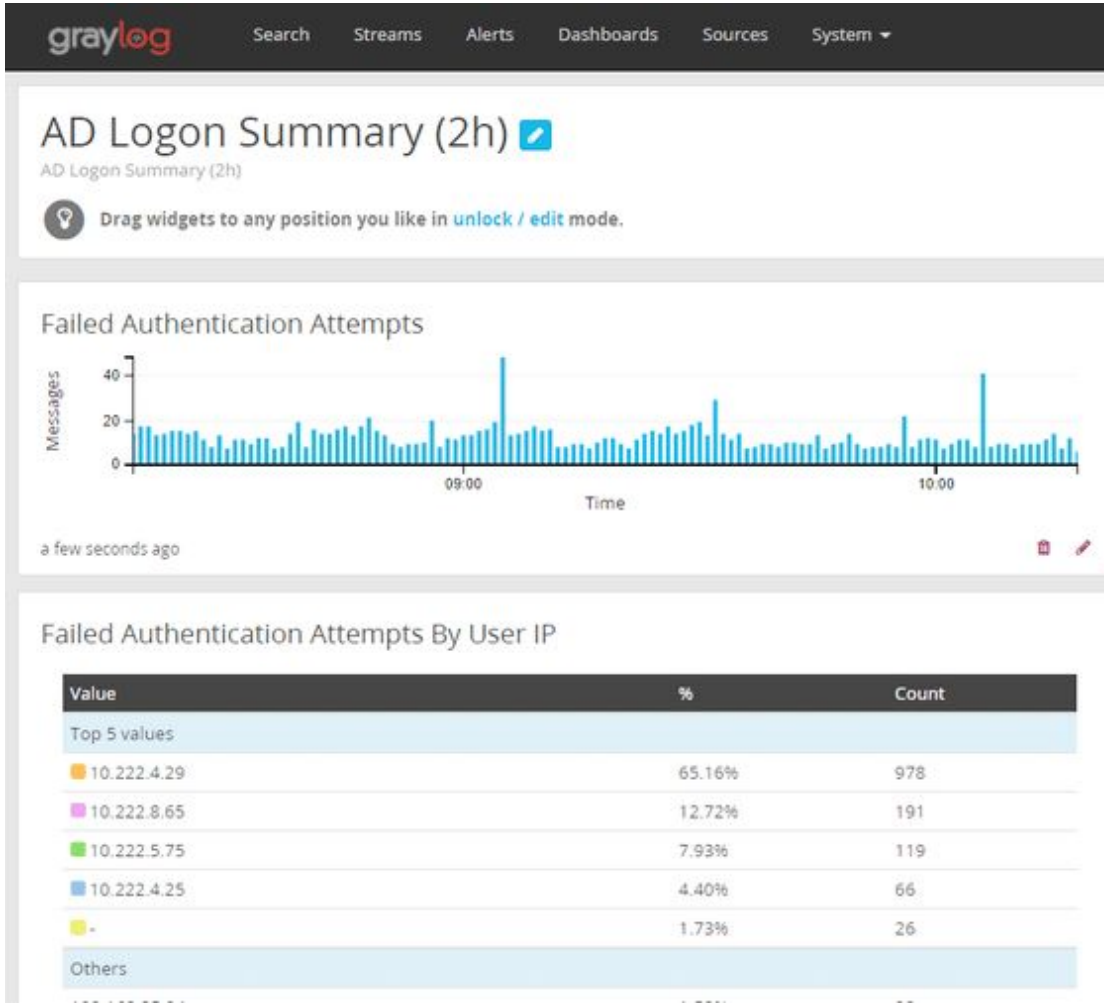
Güvenilirlik: Supervisord, bir işlem çöktüğünde otomatik olarak yeniden başlatabilir.

```
[program:charge-worker-worker-default]
process_name = %(program_name)s_%(process_num)02d
command=php /app/cli.php SubscriptionQueue processQueue default %(process_num)2d
numprocs=5
stopsignal = SIGTERM
stopwaitsecs = 60
autostart=true
autorestart=true
stderr_logfile=/var/log/supervisor/payment/charge-default-error.log
stdout_logfile=/var/log/supervisor/payment/charge-default-out.log



[program:charge-worker-worker-default]
process_name = %(program_name)s_%(process_num)02d
command=php /app/cli.php SubscriptionQueue processQueue merchant_1 %(process_num)2d
numprocs=5
stopsignal = SIGTERM
stopwaitsecs = 60
autostart=true
autorestart=true
stderr_logfile=/var/log/supervisor/payment/charge-default-error.log
stdout_logfile=/var/log/supervisor/payment/charge-default-out.log


[program:charge-worker-worker-default]
process_name = %(program_name)s_%(process_num)02d
command=php /app/cli.php SubscriptionQueue processQueue merchant_3 %(process_num)2d
numprocs=15
stopsignal = SIGTERM
stopwaitsecs = 60
autostart=true
autorestart=true
stderr_logfile=/var/log/supervisor/payment/charge-default-error.log
stdout_logfile=/var/log/supervisor/payment/charge-default-out.log
```

Graylog

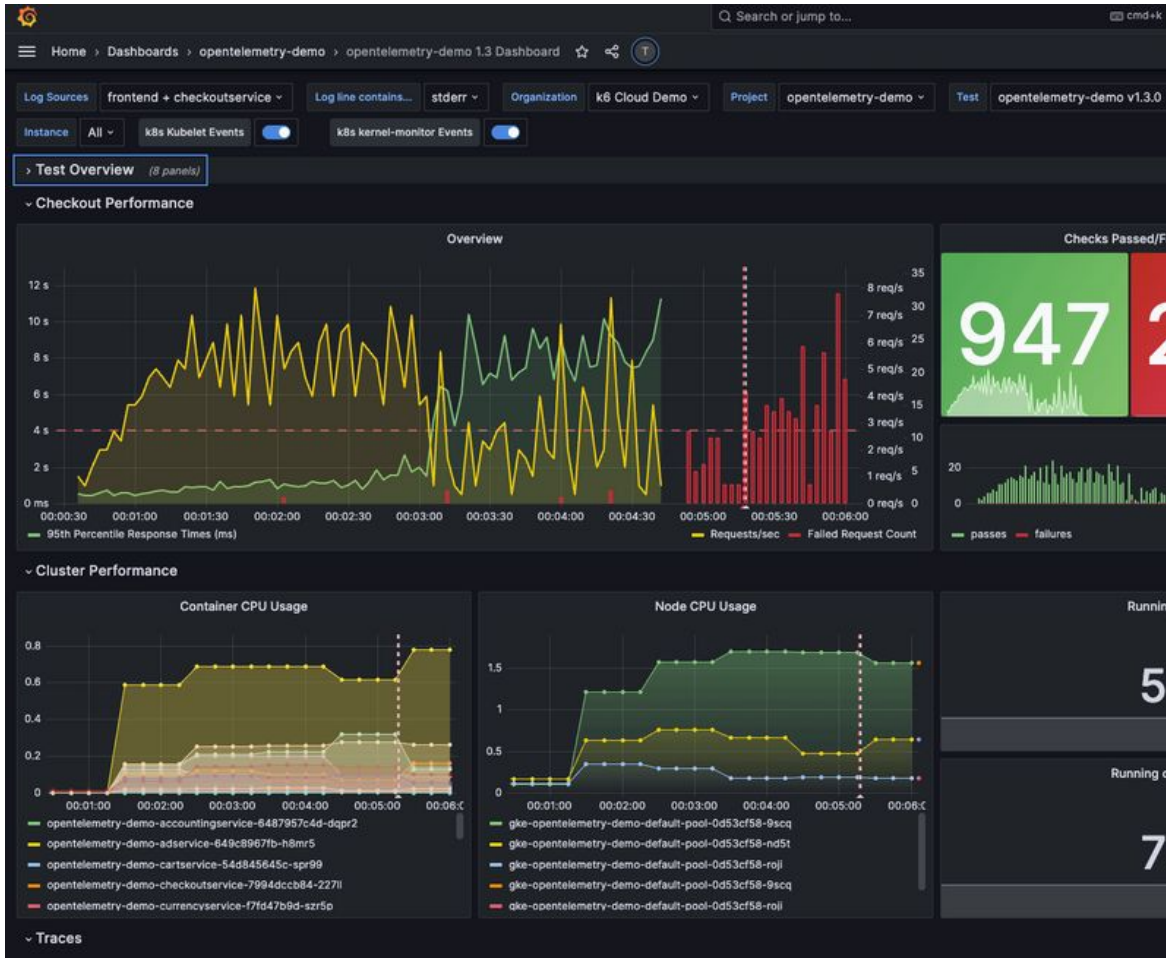
Graylog, açık kaynaklı, merkezi bir log yönetimi platformudur. Uygulamalar, sunucular, ağ aygıtları ve daha fazlasından günlükleri tek bir yerde toplayabilir, analiz edebilir ve arşivleyebilir. Bu sayede:

- Sorunları daha hızlı teşhis etmenize,
- Altyapınızı optimize etmenize
- Güvenliğinizi geliştirmenize
- Belirli senaryolar için alarmlar üretmeye
- Uygulama - web trafiğini analiz etmeye yardımcı olur.



Grafana

Grafana, açık kaynaklı bir analitik ve etkileşimli görselleştirme web uygulamasıdır. Desteklenen veri kaynaklarına bağlandığında, web için çizelgeler, grafikler ve uyarılar üretebilir.



- **Verilerinize dair daha iyi bir görüş açısı:** Verilerinizi görselleştirerek, bunları daha iyi anlayabilir ve trendleri ve modelleri belirleyebilirsiniz.

- **Daha hızlı sorun teşhisi:** Verilerinizdeki anomalileri ve sorunları daha hızlı bir şekilde tespit edebilirsiniz.

- **Geliştirilmiş altyapı performansı:** Altyapınızın performansını izleyerek ve darboğazları belirleyerek altyapınızı optimize edebilirsiniz.

TEŞEKKÜRLER

TEKNASYON

Bana Ulaşmak İsterseniz

@beytullah

@beytullah

@beytullah