ELK Vision SaaS

Technical Documentation

Enterprise Log Monitoring & Analytics Platform

Project: Big Data & Frameworks
Version: 1.0.0
Date: January 4, 2026

Django 4.2	**Next.js 14**	**ELK 8.11**	**Docker**
Backend	Frontend	Logging	Infrastructure

Contents

1 Introduction **3**

1.1 Project Context . . . 3
1.2 Objectives . . . 3
1.3 Scenario . . . 3

2 Architecture **4**

2.1 Global Architecture . . . 4
2.2 Data Flow . . . 4
2.3 Component Details . . . 4
2.3.1 Backend (Django) . . . 4
2.3.2 Frontend (Next.js) . . . 4
2.3.3 ELK Stack . . . 5
2.4 Docker Services . . . 5

3 Technical Specifications **6**

3.1 Technology Stack . . . 6
3.2 Justifications . . . 6
3.3 System Requirements . . . 6
3.4 Data Models . . . 6
3.4.1 MongoDB: Log Metadata . . . 6
3.4.2 Elasticsearch: Log Document . . . 6
3.4.3 Redis Structures . . . 7

4 Implementation **8**

4.1 ELK Configuration . . . 8
4.1.1 Logstash Pipeline . . . 8
4.2 Code Extracts . . . 8
4.2.1 WebSocket Consumer (Python) . . . 8
4.2.2 WebSocket Hook (TypeScript) . . . 8
4.3 Tests and Validation . . . 9
4.3.1 Test Coverage . . . 9
4.3.2 Performance Metrics . . . 9
4.4 Difficulties and Solutions . . . 9

5 Installation & Deployment **10**

5.1 Quick Start . . . 10
5.2 Environment Variables . . . 10
5.3 Service URLs . . . 10
5.4 Production Deployment . . . 10
5.5 Troubleshooting . . . 10

6 User Guide **11**

6.1 Getting Started 11
6.2 Dashboard 11
6.3 Uploading Logs 11
6.4 Search & Filter 12
6.5 Real-Time Logs 12
6.6 Analytics & Visualization 13
6.7 Monitoring 14

7 Conclusion & Perspectives **15**
7.1 Summary 15
7.2 Key Achievements 15
7.3 Improvement Perspectives 15
7.3.1 Short-Term 15
7.3.2 Medium-Term 15
7.3.3 Long-Term 15
7.4 References 15

1. Introduction

1.1 Project Context

ELK Vision SaaS is an enterprise-grade log monitoring platform that enables organizations to centralize, monitor, and analyze logs in real-time. Built on the ELK Stack (Elasticsearch, Logstash, Kibana) with modern web technologies, it provides scalable log aggregation, search, visualization, and alerting.

1.2 Objectives

- **Real-Time Streaming**: Sub-second latency via WebSocket and Redis Pub/Sub
- **Multi-Source Ingestion**: TCP, UDP, HTTP, and file upload support
- **Advanced Search**: Elasticsearch-powered full-text search with filtering
- **Visualization**: Built-in dashboards and Kibana integration
- **Multi-Tenant**: Secure user isolation and tenant-specific log segregation
- **Containerized**: Docker-based deployment for scalability

1.3 Scenario

A typical workflow: (1) Applications send logs to Logstash via TCP, (2) Logstash parses and enriches data, (3) Logs are indexed in Elasticsearch and published to Redis, (4) WebSocket bridge pushes logs to browsers in real-time, (5) Users filter and search logs in the dashboard.

2. Architecture

2.1 Global Architecture

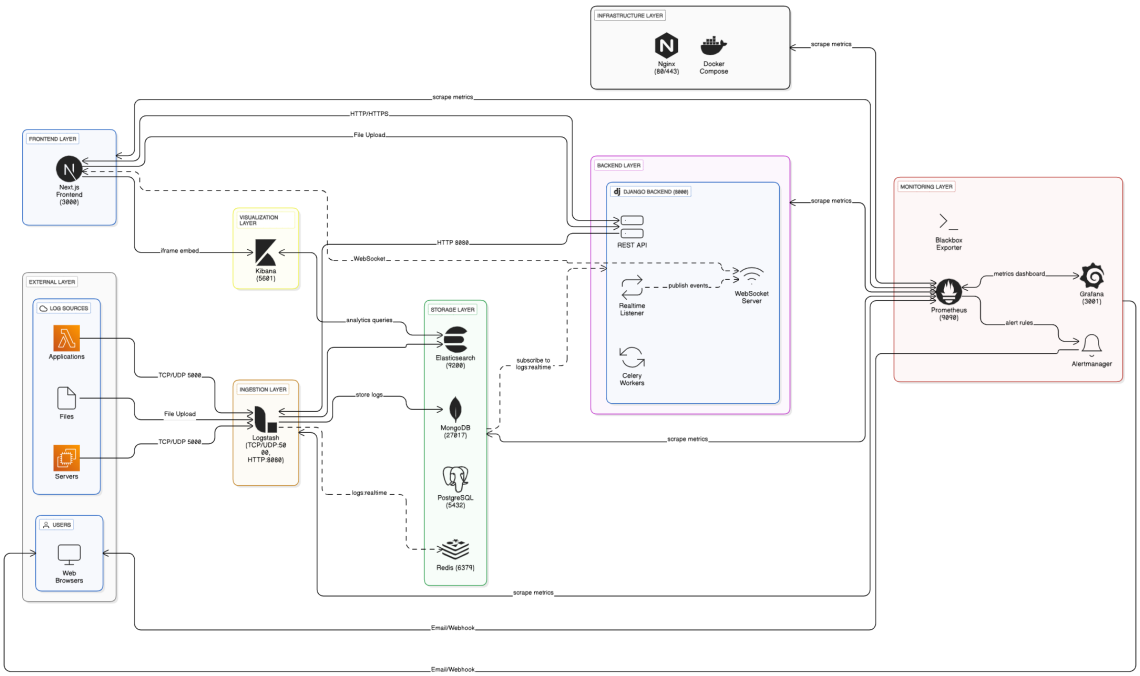

Figure 2.1: System Architecture Overview

The system follows a layered architecture:

- **Presentation**: Next.js frontend with React and WebSocket integration
- **API**: Django REST Framework with Channels for WebSocket support
- **Processing**: Logstash for log ingestion and transformation
- **Storage**: PostgreSQL (users), MongoDB (metadata), Elasticsearch (logs), Redis (cache/pubsub)

2.2 Data Flow

1. Log sources send data to Logstash (TCP:5000, UDP:5000, HTTP:8080, Beats:5044)
2. Logstash parses, transforms, and routes logs
3. Dual output: Elasticsearch (storage) + Redis Pub/Sub (real-time)
4. Real-time listener subscribes to Redis and broadcasts via WebSocket
5. Frontend receives logs with sub-second latency

2.3 Component Details

2.3.1 Backend (Django)

- **app.users**: Authentication with token-based auth
- **app.logs**: Log ingestion, search, file parsing (JSON/CSV/TXT)
- **app.dashboards**: Dashboard configurations
- **app.alerts**: Alert rules and notifications
- **api**: WebSocket consumers (LogStreamConsumer, NotificationConsumer)

2.3.2 Frontend (Next.js)

- App Router with protected routes
- Real-time components: LiveLogsView, MetricsDisplay
- Custom hooks: useWebSocket, useVisualAlerts

- Analytics with Recharts visualizations

2.3.3 ELK Stack

- **Logstash**: Multi-input pipeline with JSON parsing, GeoIP enrichment
- **Elasticsearch**: Daily indices (`logs-YYYY.MM.dd`), full-text search
- **Kibana**: Advanced visualization and custom dashboards

2.4 Docker Services

15 containerized services orchestrated via Docker Compose:

Service	Port	Purpose
frontend	3000	Next.js UI
backend	8000	Django API
postgres	5432	User data
mongodb	27017	Log metadata
redis	6379	Cache/PubSub
elasticsearch	9200	Log storage
logstash	5000	Ingestion
kibana	5601	Visualization
celery_worker	-	Async tasks
realtime_listener	-	Redis-WebSocket bridge
nginx	80/443	Reverse proxy
prometheus	9090	Metrics
grafana	3001	Dashboards

Table 2.1: Docker Services

3. Technical Specifications

3.1 Technology Stack

Category	Technology	Version
4*Backend	Python	3.11+
	Django	4.2 LTS
	Django REST Framework	3.14
	Django Channels	4.0
4*Frontend	Node.js	20 LTS
	Next.js	14.0
	React	18
	TypeScript	5
4*Databases	Elasticsearch	8.11
	PostgreSQL	15
	MongoDB	7.0
	Redis	7
3*Infrastructure	Docker	24+
	Nginx	1.25
	Logstash/Kibana	8.11

Table 3.1: Technology Stack

3.2 Justifications

- **Django**: Batteries-included framework with built-in auth, ORM, and excellent WebSocket support via Channels
- **Next.js**: React framework with App Router, SSR, and optimized performance
- **Multi-DB**: Elasticsearch for search, PostgreSQL for ACID transactions, MongoDB for flexible documents, Redis for real-time
- **Docker**: Consistent environments and easy horizontal scaling

3.3 System Requirements

Minimum: 4 CPU cores, 8 GB RAM, 50 GB disk. **Recommended**: 8+ cores, 16 GB RAM, 100 GB SSD, Ubuntu 22.04+.

3.4 Data Models

3.4.1 MongoDB: Log Metadata

```
{
  "upload_id": "uuid",
  "task_id": "celery-task-id",
  "tenant_id": "tenant_123",
  "user_id": 42,
  "filename": "app.log",
  "status": "success|pending|failed",
  "log_count": 1500,
  "created_at": "2026-01-04T12:00:00Z"
}
```

3.4.2 Elasticsearch: Log Document

```
{
  "@timestamp": "2026-01-04T12:00:00Z",
  "level": "error",
  "message": "Database connection failed",
  "source": "backend-api",
  "user_id": 42,
  "geoip": { "city": "New York", "location": {...} }
}
```

3.4.3 Redis Structures

- **Pub/Sub**: `logs:realtime` channel for streaming
- **Cache**: Session and dashboard config storage
- **Celery**: Task queue and results

4. Implementation

4.1 ELK Configuration

4.1.1 Logstash Pipeline

```
input {
  tcp { port => 5000; codec => json_lines }
  http { port => 8080; codec => json }
}
filter {
  json { source => "message"; target => "parsed" }
  date { match => ["[parsed][timestamp]", "ISO8601"] }
  mutate { lowercase => ["[parsed][level]"] }
  geoip { source => "[parsed][ip]" }
}
output {
  elasticsearch {
    hosts => ["elasticsearch:9200"]
    index => "logs-%{+YYYY.MM.dd}"
  }
  redis {
    host => "redis"; data_type => "channel"
    key => "logs:realtime"; codec => json
  }
}
```

4.2 Code Extracts

4.2.1 WebSocket Consumer (Python)

```
class LogStreamConsumer(AsyncWebsocketConsumer):
    async def connect(self):
        self.user = self.scope.get("user")
        if not self.user.is_authenticated:
            await self.close(code=4001); return
        self.group_name = f"logs_{self.user.id}"
        await self.channel_layer.group_add(
            self.group_name, self.channel_name)
        await self.accept()

    async def new_log(self, event):
        if self._matches_filters(event.get("log")):
            await self.send(json.dumps({
                "type": "new_log", "data": event["log"]
            }))
```

4.2.2 WebSocket Hook (TypeScript)

```
export function useWebSocket(endpoint: string) {
  const [isConnected, setIsConnected] = useState(false);
  const ws = useRef<WebSocket | null>(null);

  useEffect(() => {
    ws.current = new WebSocket(`${WS_URL}${endpoint}`);
    ws.current.onopen = () => setIsConnected(true);
    ws.current.onclose = () => setIsConnected(false);
    return () => ws.current?.close();
  }, [endpoint]);

  return { isConnected, ws };
}
```

4.3 Tests and Validation

4.3.1 Test Coverage

Module	Coverage	Tests
Backend (Django)	80%	73
Frontend (React)	78%	36

4.3.2 Performance Metrics

- Log ingestion: 10,000+ logs/second
- WebSocket latency: <500ms
- Search response: <200ms typical
- Concurrent users: 100+ tested

4.4 Difficulties and Solutions

Challenge	Solution
WebSocket stability	Exponential backoff reconnection, heartbeat pings
High log latency (3-5s)	Replaced polling with Redis Pub/Sub (<500ms)
ES memory issues	Reduced heap to 512MB dev, documented 4GB+ prod
Multi-DB consistency	Eventual consistency with compensation

Table 4.1: Challenges and Solutions

5. Installation & Deployment

5.1 Quick Start

```
# Clone and configure
git clone https://github.com/YasserZr/elk-vision-saas.git
cd elk-vision-saas
cp .env.example .env

# Start all services
docker compose up -d

# Initialize database
docker compose exec backend python manage.py migrate
docker compose exec backend python manage.py createsuperuser
```

5.2 Environment Variables

Variable	Description
SECRET_KEY	Django secret key
POSTGRES_PASSWORD	PostgreSQL password
MONGO_PASSWORD	MongoDB password
REDIS_PASSWORD	Redis password
ELASTICSEARCH_PASSWORD	Elasticsearch password
NEXT_PUBLIC_API_URL	Backend API URL
NEXT_PUBLIC_WS_URL	WebSocket URL

Table 5.1: Key Environment Variables

5.3 Service URLs

- Frontend: `http://localhost:3000`
- Backend API: `http://localhost:8000`
- Kibana: `http://localhost:5601`
- Grafana: `http://localhost:3001`
- Prometheus: `http://localhost:9090`
- Flower: `http://localhost:5555`

5.4 Production Deployment

```
# Build and start production
docker compose -f docker-compose.yml \
    -f docker-compose.prod.yml up -d

# Scale backend
docker compose up -d --scale backend=3
```

Security Checklist: Change all passwords, set DEBUG=False, configure HTTPS, enable ES security, restrict DB access, set up backups.

5.5 Troubleshooting

- **Services fail**: Check Docker daemon, port availability
- **DB connection**: Verify container health with `docker compose ps`
- **ES unhealthy**: Increase `ES_JAVA_OPTS` memory
- **WebSocket fails**: Check Redis and Nginx proxy config

6. User Guide

6.1 Getting Started

1. Register at `/register` (username, email, password)
2. Login at `/login`
3. Access dashboard at `/dashboard`

6.2 Dashboard

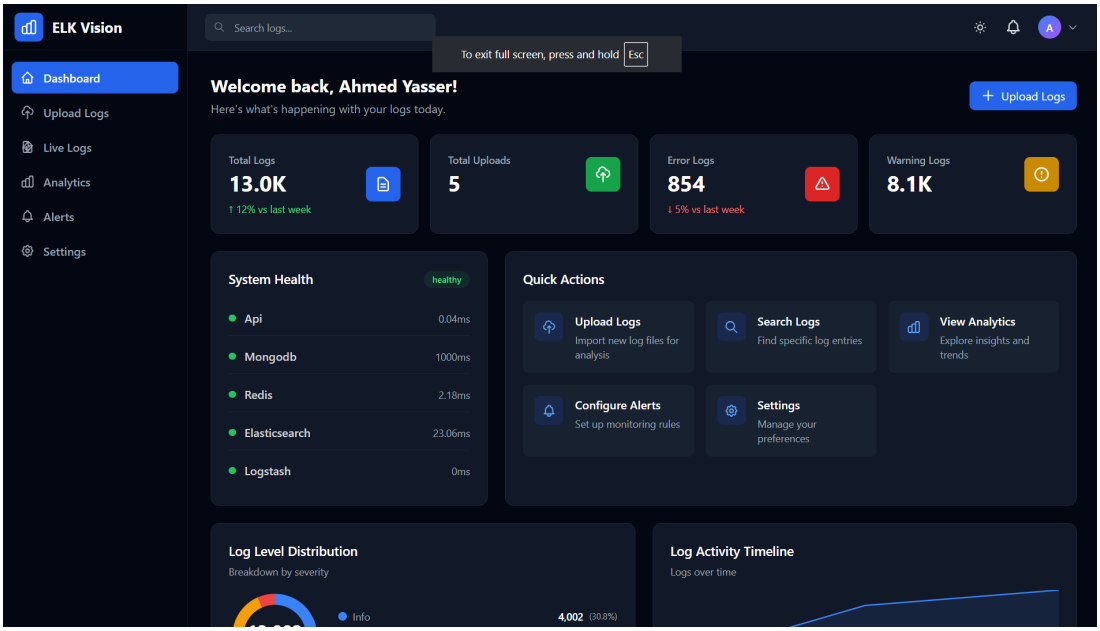

Figure 6.1: Main Dashboard

Components: KPI cards, log volume chart, level distribution, recent logs table.

6.3 Uploading Logs

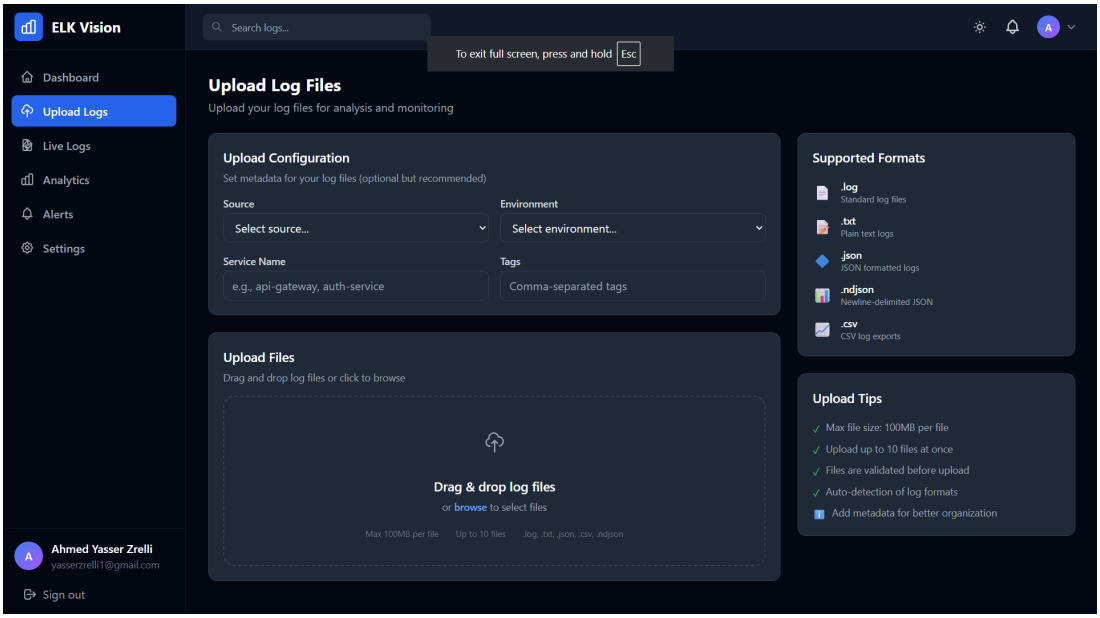

Figure 6.2: File Upload Interface

Supported formats: JSON, CSV, TXT. Drag-drop or browse, configure source/environment, monitor progress.

6.4 Search & Filter

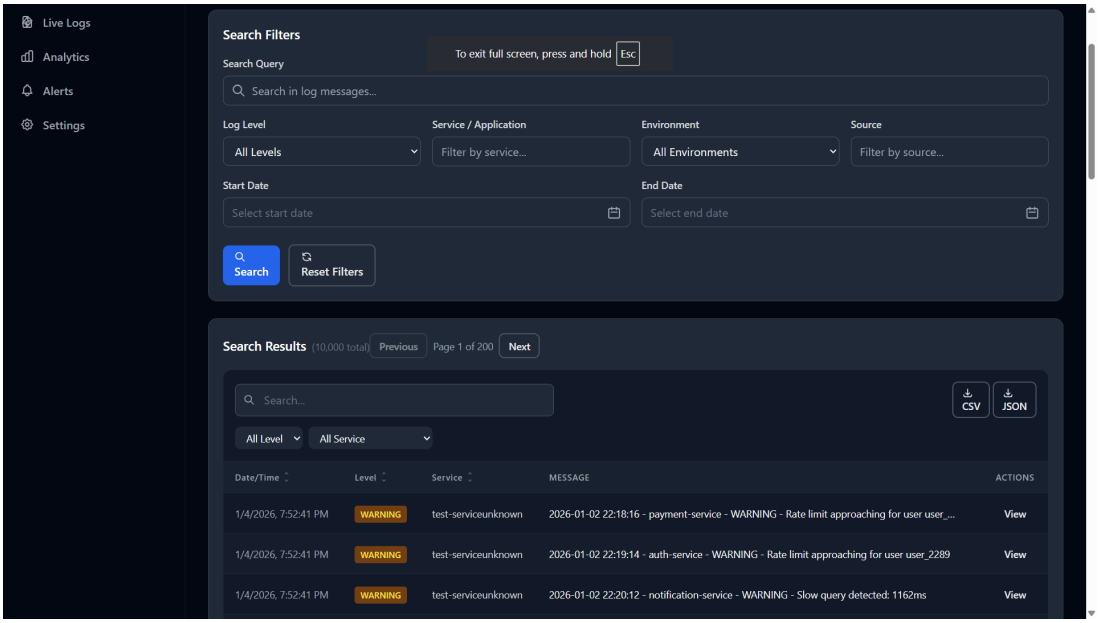

Figure 6.3: Search Interface

Filters: query text, level, source, date range, environment. Use quotes for exact phrases, wildcards for patterns.

6.5 Real-Time Logs

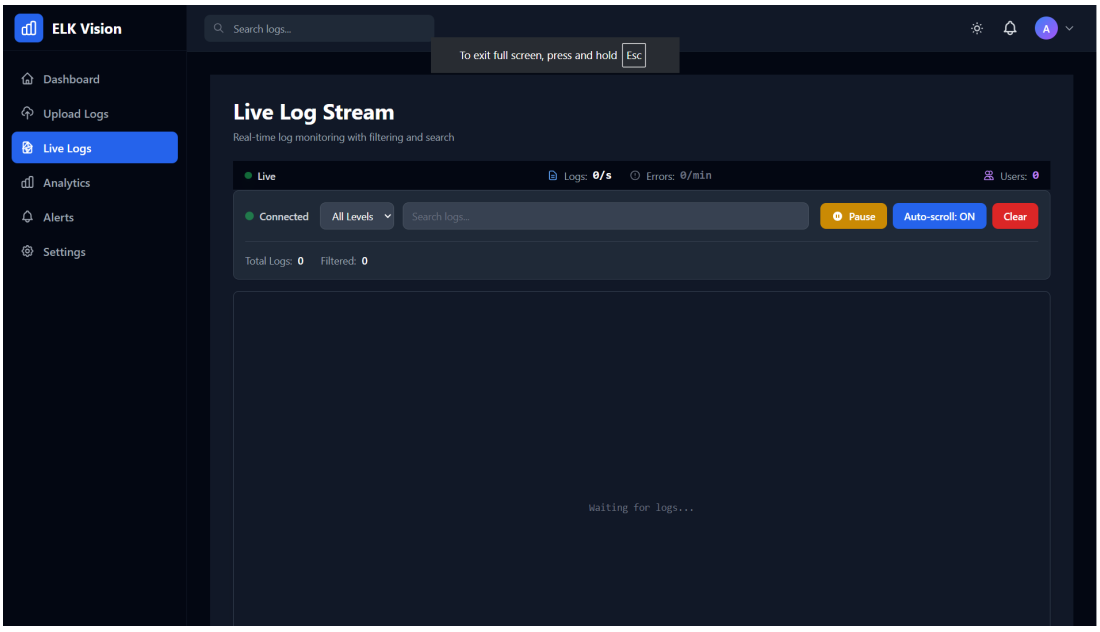

Figure 6.4: Live Log Streaming

Features: <1s latency, real-time filters, pause with Spacebar, live metrics (logs/sec, errors/min).

6.6 Analytics & Visualization

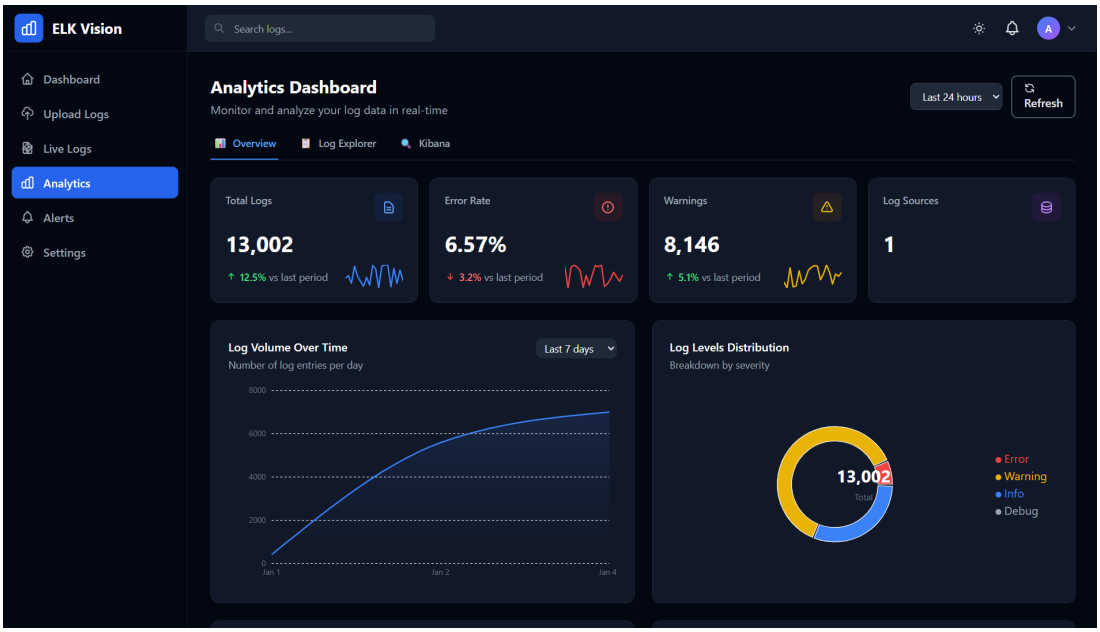

Figure 6.5: Analytics Dashboard

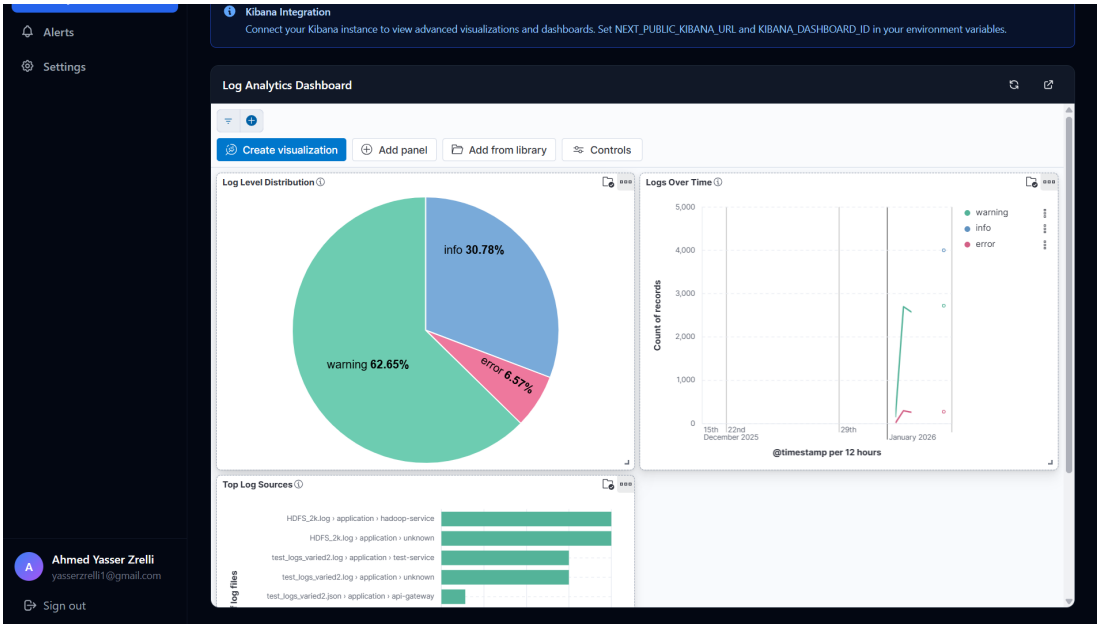

Figure 6.6: Kibana Integration

6.7 Monitoring

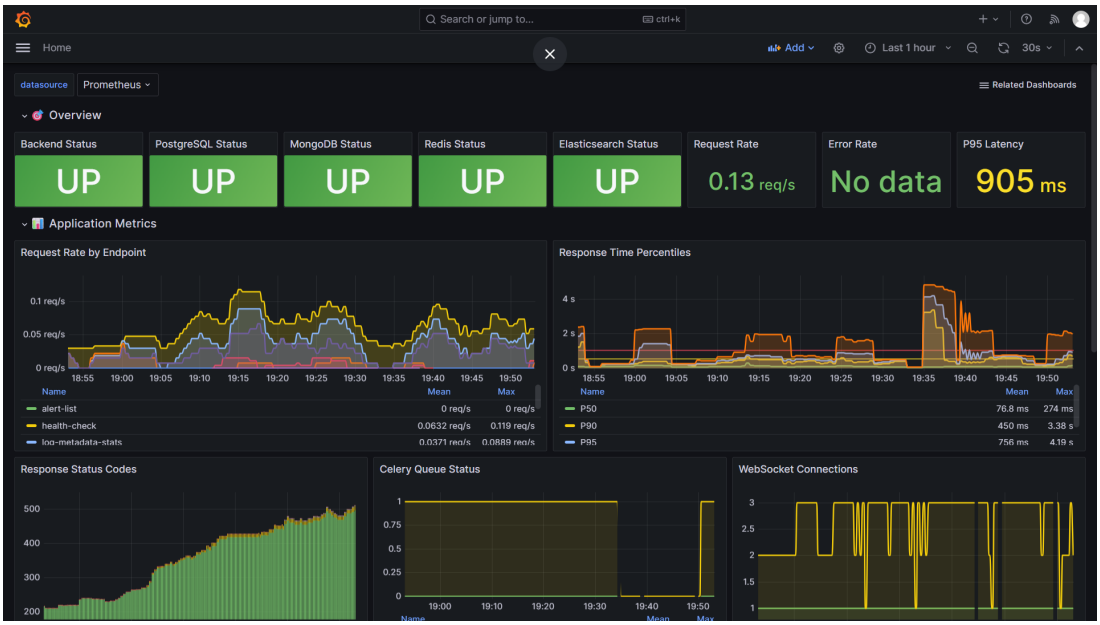

Figure 6.7: Grafana Monitoring

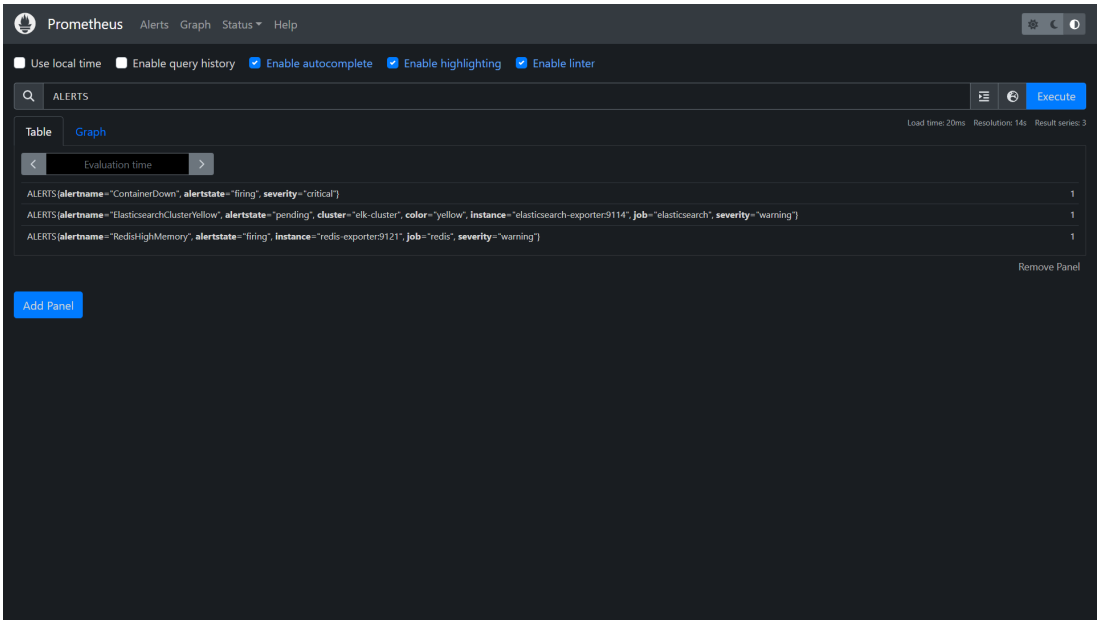

Figure 6.8: Prometheus Alerts

7. Conclusion & Perspectives

7.1 Summary

ELK Vision SaaS successfully delivers an enterprise-grade log management platform with:

- Real-time log streaming with sub-second latency
- Multi-source ingestion via TCP, UDP, HTTP, and file upload
- Advanced Elasticsearch-powered search
- Modern React/Next.js interface
- Containerized deployment with 15+ Docker services
- Comprehensive monitoring with Prometheus/Grafana

7.2 Key Achievements

- Integrated 4 databases optimized for their workloads
- Achieved 80% test coverage across backend and frontend
- Implemented WebSocket real-time with Redis Pub/Sub bridge
- Created production-ready Docker Compose configuration

7.3 Improvement Perspectives

7.3.1 Short-Term

- Email/Slack alert notifications
- Saved searches and search templates
- Dashboard export (PDF/image)

7.3.2 Medium-Term

- ML-based anomaly detection
- Log clustering and pattern recognition
- Mobile-responsive UI

7.3.3 Long-Term

- Multi-tenancy with RBAC
- Kubernetes deployment with Helm
- Distributed tracing integration

7.4 References

- Django: https://docs.djangoproject.com/
- Next.js: https://nextjs.org/docs
- Elasticsearch: https://www.elastic.co/guide/
- Project: https://github.com/YasserZr/elk-vision-saas

Document generated: January 4, 2026