ТЗ: Единый Production сервер для Legal RAG

🎯 Цель

Создать единую production инфраструктуру для Legal RAG системы с поддержкой демо, trial и платящих клиентов на одном сервере с умным управлением ресурсами и токенами OpenAI.

🏗️ Техническая архитектура

Целевые характеристики сервера:

- **CPU**: 8 vCPU (минимум 4 vCPU)
- **RAM**: 32GB (минимум 16GB)
- **SSD**: 200GB (минимум 100GB)
- **Сеть**: 1 Гбит/с
- **Платформа**: Яндекс.Облако / VK Cloud / любое облако

Поддержка нагрузки:

- **Одновременных пользователей**: 200+
- **Клиентов**: 100-500 организаций
- **Запросов в час**: 1,000-5,000
- **Документов в системе**: 100,000+

📚 Компоненты системы

1. Backend (FastAPI)

yaml

```yaml
services:
  backend:
    image: legal-rag:production
    deploy:
      replicas: 3
    environment:
      - ENVIRONMENT=production
      - DATABASE_URL=postgresql://user:pass@postgres:5432/legal_rag_prod
      - REDIS_URL=redis://redis:6379
      - OPENAI_API_KEY=${OPENAI_API_KEY}
      - CHROMA_DB_PATH=/data/chroma_db
    volumes:
      - ./uploads:/app/uploads
      - ./chroma_db:/data/chroma_db
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/api/health"]
      interval: 30s
      timeout: 10s
      retries: 3
```

2. База данных (PostgreSQL)

yaml

```yaml
  postgres:
    image: postgres:15-alpine
    environment:
      - POSTGRES_DB=legal_rag_prod
      - POSTGRES_USER=legal_rag_user
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_SHARED_PRELOAD_LIBRARIES=pg_stat_statements
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
    command: >
      postgres
      -c max_connections=200
      -c shared_buffers=256MB
      -c effective_cache_size=1GB
      -c maintenance_work_mem=64MB
      -c checkpoint_completion_target=0.9
      -c wal_buffers=16MB
      -c default_statistics_target=100
```

3. Redis (кэширование + rate limiting)

yaml

```yaml
redis:
  image: redis:7-alpine
  command: redis-server --maxmemory 1gb --maxmemory-policy allkeys-lru
  volumes:
    - redis_data:/data
```

4. Nginx (reverse proxy + load balancer)

yaml

```yaml
nginx:
  image: nginx:alpine
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./nginx.prod.conf:/etc/nginx/nginx.conf
    - ./ssl:/etc/nginx/ssl
  depends_on:
    - backend
```

🔐 Система лимитов и тарификации

1. Модель тарифов

python

```python
# backend/models/billing.py
from enum import Enum

class SubscriptionTier(Enum):
    DEMO = "demo"
    BASIC = "basic"
    PROFESSIONAL = "professional"
    ENTERPRISE = "enterprise"

class TierLimits:
    DEMO = {
        "queries_per_day": 10,
        "queries_per_month": 100,
        "documents_max": 5,
        "users_max": 1,
        "ai_model": "gpt-3.5-turbo",
        "max_tokens_per_query": 500,
        "file_size_max_mb": 5,
        "storage_max_gb": 0.1,
    }

    BASIC = {
        "queries_per_day": 50,
        "queries_per_month": 500,
        "documents_max": 100,
        "users_max": 3,
        "ai_model": "gpt-3.5-turbo",
        "max_tokens_per_query": 1000,
        "file_size_max_mb": 10,
        "storage_max_gb": 1,
        "price_per_month": 5000,
    }

    PROFESSIONAL = {
        "queries_per_day": 200,
        "queries_per_month": 2000,
        "documents_max": 500,
        "users_max": 10,
        "ai_model": "gpt-4",
        "max_tokens_per_query": 2000,
        "file_size_max_mb": 50,
        "storage_max_gb": 5,
        "price_per_month": 15000,
    }

    ENTERPRISE = {
```

```
    "queries_per_day": 1000,
    "queries_per_month": 10000,
    "documents_max": 2000,
    "users_max": 50,
    "ai_model": "gpt-4",
    "max_tokens_per_query": 4000,
    "file_size_max_mb": 100,
    "storage_max_gb": 20,
    "price_per_month": 50000,
}
```

2. Middleware для контроля лимитов

python

```python
# backend/middleware/rate_limiter.py
from fastapi import HTTPException, Depends
import redis
import json
from datetime import datetime, timedelta

class RateLimiter:
    def __init__(self):
        self.redis = redis.Redis(host='redis', port=6379, db=0)

    async def check_daily_limit(self, user_id: str, org_id: str):
        """Проверка дневного лимита запросов"""
        key = f"daily_queries:{org_id}:{datetime.now().strftime('%Y-%m-%d')}"
        current_count = self.redis.get(key) or 0

        org = await get_organization(org_id)
        daily_limit = TierLimits.__dict__[org.subscription_tier.upper()]["queries_per_day"]

        if int(current_count) >= daily_limit:
            raise HTTPException(
                status_code=429,
                detail=f"Превышен дневной лимит запросов ({daily_limit})"
            )

        # Увеличиваем счетчик
        pipe = self.redis.pipeline()
        pipe.incr(key)
        pipe.expire(key, 86400)  # 24 часа
        pipe.execute()

    async def check_monthly_limit(self, org_id: str):
        """Проверка месячного лимита"""
        key = f"monthly_queries:{org_id}:{datetime.now().strftime('%Y-%m')}"
        current_count = self.redis.get(key) or 0

        org = await get_organization(org_id)
        monthly_limit = TierLimits.__dict__[org.subscription_tier.upper()]["queries_per_month"]

        if int(current_count) >= monthly_limit:
            raise HTTPException(
                status_code=429,
                detail=f"Превышен месячный лимит запросов ({monthly_limit})"
            )
```

3. Умное управление токенами OpenAI

python

```python
# backend/services/ai_service.py
class SmartAIService:
    def __init__(self):
        self.openai_client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))
        self.redis = redis.Redis(host='redis')

    async def choose_model(self, tier: str, query_complexity: int):
        """Выбор модели на основе тарифа и сложности"""
        if tier == "DEMO" or tier == "BASIC":
            return "gpt-3.5-turbo"
        elif tier == "PROFESSIONAL":
            return "gpt-4" if query_complexity > 7 else "gpt-3.5-turbo"
        else:  # ENTERPRISE
            return "gpt-4"

    async def get_cached_response(self, query_hash: str):
        """Проверка кэша перед обращением к OpenAI"""
        cached = await self.redis.get(f"ai_cache:{query_hash}")
        if cached:
            return json.loads(cached)
        return None

    async def cache_response(self, query_hash: str, response: dict):
        """Кэширование ответа на 24 часа"""
        await self.redis.setex(
            f"ai_cache:{query_hash}",
            86400,  # 24 часа
            json.dumps(response)
        )

    async def track_token_usage(self, org_id: str, tokens_used: int, model: str):
        """Отслеживание использования токенов"""
        cost_per_token = {
            "gpt-3.5-turbo": 0.0005,  # за 1K токенов
            "gpt-4": 0.01
        }

        cost = (tokens_used / 1000) * cost_per_token.get(model, 0.001)

        # Обновляем статистику в БД
        await self.db.execute("""
            UPDATE organizations
            SET
                tokens_used_month = tokens_used_month + %s,
                ai_cost_month = ai_cost_month + %s
```

```
        WHERE id = %s
    """, [tokens_used, cost, org_id])
```

🗄️ Структура базы данных

Дополнительные таблицы для биллинга:

sql

```sql
-- Расширение таблицы organizations
ALTER TABLE organizations ADD COLUMN IF NOT EXISTS subscription_tier VARCHAR(20) DEFAULT 'demo';
ALTER TABLE organizations ADD COLUMN IF NOT EXISTS subscription_start DATE;
ALTER TABLE organizations ADD COLUMN IF NOT EXISTS subscription_end DATE;
ALTER TABLE organizations ADD COLUMN IF NOT EXISTS tokens_used_month INTEGER DEFAULT 0;
ALTER TABLE organizations ADD COLUMN IF NOT EXISTS ai_cost_month DECIMAL(10,2) DEFAULT 0;
ALTER TABLE organizations ADD COLUMN IF NOT EXISTS queries_used_month INTEGER DEFAULT 0;
ALTER TABLE organizations ADD COLUMN IF NOT EXISTS storage_used_gb DECIMAL(5,2) DEFAULT 0;

-- Таблица для отслеживания использования
CREATE TABLE IF NOT EXISTS usage_logs (
    id SERIAL PRIMARY KEY,
    organization_id VARCHAR(100) NOT NULL,
    user_id VARCHAR(100) NOT NULL,
    action_type VARCHAR(50) NOT NULL, -- 'ai_query', 'document_upload', 'search'
    tokens_used INTEGER DEFAULT 0,
    cost DECIMAL(8,4) DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    metadata JSONB
);

-- Индексы для быстрого поиска
CREATE INDEX IF NOT EXISTS idx_usage_logs_org_date ON usage_logs(organization_id, created_at);
CREATE INDEX IF NOT EXISTS idx_usage_logs_user_date ON usage_logs(user_id, created_at);

-- Таблица для тарифных планов
CREATE TABLE IF NOT EXISTS subscription_plans (
    id SERIAL PRIMARY KEY,
    tier VARCHAR(20) UNIQUE NOT NULL,
    name VARCHAR(100) NOT NULL,
    price_monthly INTEGER NOT NULL,
    queries_per_month INTEGER NOT NULL,
    documents_max INTEGER NOT NULL,
    users_max INTEGER NOT NULL,
    storage_gb INTEGER NOT NULL,
    features JSONB,
    is_active BOOLEAN DEFAULT true
);

-- Заполнение базовых тарифов
INSERT INTO subscription_plans (tier, name, price_monthly, queries_per_month, documents_max, users_max, st
('demo', 'Демо', 0, 100, 5, 1, 1, '{"ai_model": "gpt-3.5-turbo", "support": "community"}'),
('basic', 'Базовый', 5000, 500, 100, 3, 5, '{"ai_model": "gpt-3.5-turbo", "support": "email"}'),
('professional', 'Профессиональный', 15000, 2000, 500, 10, 20, '{"ai_model": "gpt-4", "support": "priority"}'),
('enterprise', 'Корпоративный', 50000, 10000, 2000, 50, 100, '{"ai_model": "gpt-4", "support": "24/7", "custom_
```

🔧 Конфигурация Nginx

nginx.prod.conf:

nginx

```
events {
    worker_connections 1024;
}

http {
    upstream backend {
        least_conn;
        server backend:8000 max_fails=3 fail_timeout=30s;
    }

    # Rate limiting
    limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;
    limit_req_zone $binary_remote_addr zone=auth:10m rate=5r/s;

    server {
        listen 80;
        server_name your-domain.com;

        # Redirect HTTP to HTTPS
        return 301 https://$server_name$request_uri;
    }

    server {
        listen 443 ssl http2;
        server_name your-domain.com;

        ssl_certificate /etc/nginx/ssl/cert.pem;
        ssl_certificate_key /etc/nginx/ssl/key.pem;

        # Security headers
        add_header X-Frame-Options DENY;
        add_header X-Content-Type-Options nosniff;
        add_header X-XSS-Protection "1; mode=block";

        # File upload limits
        client_max_body_size 100M;

        # API routes
        location /api/ {
            limit_req zone=api burst=20 nodelay;
            proxy_pass http://backend;
            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;
```

```
        # Timeouts
        proxy_connect_timeout 30s;
        proxy_send_timeout 30s;
        proxy_read_timeout 60s;
    }

    # Auth routes with stricter limits
    location /api/auth/ {
        limit_req zone=auth burst=5 nodelay;
        proxy_pass http://backend;
        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;
    }

    # Static files (if serving frontend)
    location / {
        root /usr/share/nginx/html;
        try_files $uri $uri/ /index.html;
    }

    # Health check
    location /health {
        access_log off;
        return 200 "healthy\n";
        add_header Content-Type text/plain;
    }
  }
}
```

📊 Мониторинг и логирование

1. Health checks

python

```python
# backend/api/health.py
@app.get("/api/health")
async def health_check():
    checks = {
        "database": await check_database_connection(),
        "redis": await check_redis_connection(),
        "disk_space": await check_disk_space(),
        "memory": await check_memory_usage(),
        "openai": await check_openai_connection(),
    }

    status = "healthy" if all(checks.values()) else "unhealthy"

    return {
        "status": status,
        "timestamp": datetime.utcnow().isoformat(),
        "checks": checks,
        "version": "1.0.0"
    }
```

2. Метрики производительности

python

```python
# backend/middleware/metrics.py
from prometheus_client import Counter, Histogram, Gauge

# Метрики
REQUEST_COUNT = Counter('http_requests_total', 'Total requests', ['method', 'endpoint', 'status'])
REQUEST_DURATION = Histogram('http_request_duration_seconds', 'Request duration')
ACTIVE_USERS = Gauge('active_users_total', 'Active users')
AI_REQUESTS = Counter('ai_requests_total', 'AI requests', ['model', 'organization'])
TOKEN_USAGE = Counter('openai_tokens_total', 'OpenAI tokens used', ['model', 'organization'])

@app.middleware("http")
async def metrics_middleware(request: Request, call_next):
    start_time = time.time()
    response = await call_next(request)
    duration = time.time() - start_time

    REQUEST_COUNT.labels(
        method=request.method,
        endpoint=request.url.path,
        status=response.status_code
    ).inc()

    REQUEST_DURATION.observe(duration)

    return response
```

🚀 Развертывание

1. Переменные окружения (.env.production)

bash

```
# Database
POSTGRES_PASSWORD=secure_production_password_here
DATABASE_URL=postgresql://legal_rag_user:${POSTGRES_PASSWORD}@postgres:5432/legal_rag_prod

# Redis
REDIS_URL=redis://redis:6379

# OpenAI
OPENAI_API_KEY=your_openai_api_key_here

# Application
ENVIRONMENT=production
SECRET_KEY=your_jwt_secret_key_here
DEBUG=False
ALLOWED_HOSTS=your-domain.com,www.your-domain.com

# Storage
UPLOAD_DIR=/app/uploads
CHROMA_DB_PATH=/data/chroma_db

# Email (for notifications)
SMTP_HOST=smtp.yandex.ru
SMTP_PORT=587
SMTP_USER=noreply@your-domain.com
SMTP_PASSWORD=your_email_password

# Monitoring
ENABLE_METRICS=true
LOG_LEVEL=INFO
SENTRY_DSN=your_sentry_dsn_here
```

2. Docker Compose для production

yaml

```yaml
# docker-compose.production.yml
version: '3.8'

services:
  backend:
    build:
      context: .
      dockerfile: Dockerfile.production
    environment:
      - ENVIRONMENT=production
    env_file:
      - .env.production
    volumes:
      - ./uploads:/app/uploads
      - ./chroma_db:/data/chroma_db
      - ./logs:/app/logs
    deploy:
      replicas: 2
      restart_policy:
        condition: on-failure
        max_attempts: 3
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/api/health"]
      interval: 30s
      timeout: 10s
      retries: 3
    depends_on:
      - postgres
      - redis

  postgres:
    image: postgres:15-alpine
    env_file:
      - .env.production
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql
      - ./backups:/backups
    command: >
      postgres
      -c max_connections=200
      -c shared_buffers=512MB
      -c effective_cache_size=2GB
    deploy:
      resources:
        limits:
```

```
        memory: 4G
      reservations:
        memory: 2G

  redis:
    image: redis:7-alpine
    command: redis-server --maxmemory 2gb --maxmemory-policy allkeys-lru
    volumes:
      - redis_data:/data
    deploy:
      resources:
        limits:
          memory: 2G

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.prod.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
      - ./static:/usr/share/nginx/html
    depends_on:
      - backend
    deploy:
      resources:
        limits:
          memory: 256M

volumes:
  postgres_data:
  redis_data:
```

📋 Чек-лист развертывания

Подготовка:

- [ ] Заказать сервер (8 vCPU, 32GB RAM, 200GB SSD)
- [ ] Настроить домен и DNS
- [ ] Получить SSL сертификат
- [ ] Создать .env.production файл
- [ ] Подготовить backup стратегию

Развертывание:

- [ ] Клонировать код на сервер
- [ ] Запустить `docker-compose -f docker-compose.production.yml up -d`
- [ ] Выполнить миграции БД
- [ ] Создать администратора
- [ ] Настроить мониторинг
- [ ] Протестировать все функции

Тестирование:

- [ ] Создать демо-аккаунт и проверить лимиты
- [ ] Создать платящий аккаунт и проверить функционал
- [ ] Протестировать загрузку документов
- [ ] Протестировать AI запросы
- [ ] Проверить систему биллинга
- [ ] Нагрузочное тестирование

Мониторинг:

- [ ] Настроить алерты на превышение лимитов
- [ ] Настроить backup БД (ежедневно)
- [ ] Настроить мониторинг ресурсов сервера
- [ ] Настроить логирование ошибок

🎯 Ожидаемые результаты

После развертывания система должна поддерживать:

- **100-500 организаций** одновременно
- **1000+ одновременных пользователей**
- **Uptime 99.9%**
- **Время отклика API < 200ms**
- **Обработка файлов до 100MB**
- **Автоматическое масштабирование** нагрузки