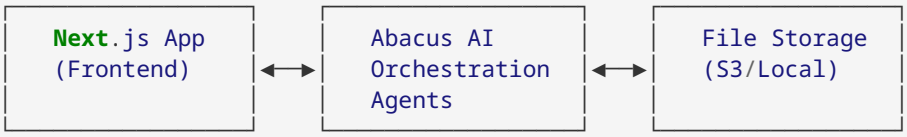
THANOS Deployment Guide

Overview

This guide covers deploying the complete THANOS system including the Next.js web application and Abacus AI orchestration agents.

Architecture



Prerequisites

System Requirements

- Node.js 18+ and Yarn
- Python 3.8+ (for agent tools)
- Database (PostgreSQL recommended)
- File storage (S3 or local filesystem)
- Abacus AI account

Services Required

- **Web Hosting**: Vercel, Netlify, or custom server
- **Database**: Supabase, PlanetScale, or self-hosted PostgreSQL
- **File Storage**: AWS S3, Google Cloud Storage, or local storage
- **AI Processing**: Abacus AI platform

Deployment Options

Option 1: Full Cloud Deployment (Recommended)

1. Web Application Deployment

Deploy to Vercel:

```
# Install Vercel CLI
npm i -g vercel

# Deploy from project root
cd web-app
vercel --prod
```

Deploy to Netlify:

```
# Install Netlify CLI
npm i -g netlify-cli

# Build and deploy
yarn build
netlify deploy --prod --dir=.next
```

Environment Variables:

```
# Database
DATABASE_URL=your_postgres_connection_string
DIRECT_URL=your_direct_postgres_url

# S3 Storage
AWS_ACCESS_KEY_ID=your_access_key
AWS_SECRET_ACCESS_KEY=your_secret_key
AWS_REGION=us-east-1
S3_BUCKET_NAME=thanos-files

# Abacus AI Integration
ABACUS_API_KEY=your_abacus_api_key
ABACUS_ORCHESTRATOR_URL=your_deployment_url
ABACUS_PROJECT_ID=your_project_id

# App Configuration
NEXTAUTH_URL=https://your-domain.com
NEXTAUTH_SECRET=your_secret_key
```

2. Database Setup

Using Supabase:

```
-- Run this SQL in Supabase SQL editor
-- Tables are created automatically by Prisma migrations
-- Just need to enable Row Level Security if needed

-- Enable RLS on tables
ALTER TABLE "Organization" ENABLE ROW LEVEL SECURITY;
ALTER TABLE "FileRecord" ENABLE ROW LEVEL SECURITY;
```

Using PlanetScale:

```
# Connect to PlanetScale
pscale connect thanos-db main

# Run Prisma migrations
cd web-app
npx prisma db push
```

3. Abacus AI Deployment

Follow the Abacus AI Deployment Guide (./agents/config/abacus-deployment.md):

1. Create Abacus AI project
2. Upload agent tools
3. Deploy SnapOrchestrator

4. Configure webhooks

Option 2: Self-Hosted Deployment

1. Server Setup

System Requirements:

- Ubuntu 20.04+ or similar
- 4GB+ RAM
- 50GB+ storage
- Docker and Docker Compose

Install Dependencies:

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Node.js 18
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs

# Install Yarn
npm install -g yarn

# Install Docker
sudo apt-get install docker.io docker-compose
sudo usermod -aG docker $USER
```

2. Database Setup

PostgreSQL with Docker:

```
# docker-compose.yml
version: '3.8'
services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: thanos
      POSTGRES_USER: thanos
      POSTGRES_PASSWORD: your_password
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped

volumes:
  postgres_data:
```

Start Database:

```
docker-compose up -d postgres
```

3. Application Deployment

Build and Start App:

```
# Clone repository
git clone your-repo-url
cd thanos-complete-system/web-app

# Install dependencies
yarn install

# Set up environment variables
cp .env.example .env.local
# Edit .env.local with your configuration

# Generate Prisma client and run migrations
npx prisma generate
npx prisma db push

# Build application
yarn build

# Start production server
yarn start
```

Process Management with PM2:

```
# Install PM2
npm install -g pm2

# Create ecosystem file
cat > ecosystem.config.js << EOF
module.exports = {
  apps: [{
    name: 'thanos-app',
    script: 'yarn start',
    cwd: './web-app',
    instances: 2,
    exec_mode: 'cluster',
    env: {
      NODE_ENV: 'production',
      PORT: 3000
    }
  }]
}
EOF

# Start with PM2
pm2 start ecosystem.config.js
pm2 save
pm2 startup
```

Option 3: Docker Deployment

Complete Docker Setup:

```dockerfile
# web-app/Dockerfile
FROM node:18-alpine AS base
WORKDIR /app

# Install dependencies
COPY package.json yarn.lock ./
RUN yarn install --frozen-lockfile

# Build application
COPY . .
RUN yarn build

FROM node:18-alpine AS runner
WORKDIR /app

ENV NODE_ENV production

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=base /app/public ./public
COPY --from=base --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=base --chown=nextjs:nodejs /app/.next/static ./.next/static

USER nextjs

EXPOSE 3000
ENV PORT 3000

CMD ["node", "server.js"]
```

Docker Compose for Full Stack:

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

services:
  app:
    build: ./web-app
    ports:
      - "3000:3000"
    environment:
      - DATABASE_URL=postgresql://thanos:password@postgres:5432/thanos
      - NEXTAUTH_URL=https://your-domain.com
    depends_on:
      - postgres
    restart: unless-stopped

  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: thanos
      POSTGRES_USER: thanos
      POSTGRES_PASSWORD: your_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - app
    restart: unless-stopped

volumes:
  postgres_data:
```

Production Configuration

1. Environment Variables

Required Variables:

```
# App
NODE_ENV=production
NEXTAUTH_URL=https://your-domain.com
NEXTAUTH_SECRET=your-32-char-secret

# Database
DATABASE_URL=your-production-db-url
DIRECT_URL=your-direct-db-url

# File Storage
AWS_ACCESS_KEY_ID=your-access-key
AWS_SECRET_ACCESS_KEY=your-secret-key
S3_BUCKET_NAME=your-production-bucket

# Abacus AI
ABACUS_API_KEY=your-production-api-key
ABACUS_ORCHESTRATOR_URL=your-production-orchestrator
```

2. SSL/TLS Setup

Using Nginx:

```
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;

    location / {
        proxy_pass http://app:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        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;
    }
}
```

3. Monitoring and Logging

Application Monitoring:

```
# Install monitoring tools
npm install -g @vercel/analytics
npm install -g @sentry/nextjs

# Set up logging
mkdir -p /var/log/thanos
chown www-data:www-data /var/log/thanos
```

Log Configuration:

```
// web-app/lib/logger.js
import winston from 'winston';

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  defaultMeta: { service: 'thanos-app' },
  transports: [
    new winston.transports.File({ filename: '/var/log/thanos/error.log', level:
'error' }),
    new winston.transports.File({ filename: '/var/log/thanos/combined.log' }),
  ],
});

if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.simple()
  }));
}

export default logger;
```

Scaling Considerations

1. Horizontal Scaling

Load Balancer Setup:

```
# kubernetes/deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: thanos-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: thanos-app
  template:
    metadata:
      labels:
        app: thanos-app
    spec:
      containers:
      - name: thanos-app
        image: your-registry/thanos-app:latest
        ports:
        - containerPort: 3000
```

2. Database Scaling

Read Replicas:

```
# Set up read replicas for better performance
DATABASE_URL=postgresql://write-user@write-db/thanos
DATABASE_READ_URL=postgresql://read-user@read-db/thanos
```

3. File Storage Scaling

CDN Setup:

```
// Configure CloudFront or similar CDN
const CDN_URL = process.env.CDN_URL || process.env.S3_BUCKET_URL;

export function getFileUrl(key) {
  return `${CDN_URL}/${key}`;
}
```

Backup and Recovery

1. Database Backups

```
#!/bin/bash
# backup-db.sh

DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/backups/database"
mkdir -p $BACKUP_DIR

pg_dump $DATABASE_URL > $BACKUP_DIR/thanos_backup_$DATE.sql

# Keep only last 7 days of backups
find $BACKUP_DIR -name "*.sql" -mtime +7 -delete
```

2. File Storage Backups

```
#!/bin/bash
# backup-files.sh

aws s3 sync s3://your-bucket s3://your-backup-bucket \
  --storage-class GLACIER
```

Security Checklist

- [ ] SSL/TLS certificates configured
- [ ] Environment variables secured
- [ ] Database connections encrypted
- [ ] API keys rotated regularly
- [ ] CORS properly configured
- [ ] Rate limiting enabled
- [ ] Input validation implemented
- [ ] File upload restrictions in place
- [ ] Regular security updates applied

Troubleshooting

Common Issues

1. **Build Failures:**
   - Check Node.js version compatibility
   - Verify environment variables
   - Clear node_modules and yarn cache

2. **Database Connection Issues:**
   - Verify connection strings
   - Check firewall rules
   - Test database connectivity

3. **File Upload Failures:**
   - Check S3 permissions
   - Verify bucket policy
   - Test file size limits

4. **Abacus AI Integration Issues:**
   - Verify API keys
   - Check deployment status
   - Review agent logs

Performance Optimization

1. **Next.js Optimizations:**
   - Enable image optimization
   - Use dynamic imports
   - Configure caching headers
   - Optimize bundle size

2. **Database Optimizations:**
   - Add appropriate indexes
   - Use connection pooling
   - Optimize queries

3. **File Processing:**
   - Implement batch processing
   - Use background jobs
   - Optimize image processing

For additional help, refer to the API Reference (API_REFERENCE.md) or contact support.