Deployment Guide

Prerequisites

System Requirements

- Docker 20.10+ with Compose plugin
- Git 2.30+
- 8GB RAM minimum (16GB recommended)
- 20GB free disk space
- Linux/macOS/Windows with WSL2

External Services

- Abacus.AI account with API key
- GitHub account (for repository access)

Local Development Setup

1. Clone the Repository

```
git clone https://github.com/nbbulk-dotcom/COSMOLOGY.git
cd COSMOLOGY
```

2. Configure Environment

```
# Copy the example environment file
cp .env.example .env

# Edit .env with your configuration
nano .env
```

Required Configuration:
- `ABACUSAI_API_KEY`: Your Abacus.AI API key
- `POSTGRES_PASSWORD`: Change from default
- `S3_ACCESS_KEY_ID` & `S3_SECRET_ACCESS_KEY`: For production S3

3. Start Services

```
# Start all services in detached mode
docker-compose up -d

# View logs
docker-compose logs -f

# Check service status
docker-compose ps
```

4. Verify Installation

```
# Check backend health
curl http://localhost:8000/health

# Access frontend
open http://localhost:3000

# View API documentation
open http://localhost:8000/docs
```

5. Run Database Migrations

```
# Enter backend container
docker-compose exec backend bash

# Run Alembic migrations
alembic upgrade head

# Exit container
exit
```

Production Deployment

Docker Compose (Single Server)

For small to medium deployments:

```
# Use production compose file
docker-compose -f docker-compose.yml up -d

# Enable automatic restarts
docker-compose up -d --force-recreate
```

Kubernetes Deployment

For scalable production deployments:

1. Build and push images to container registry
2. Deploy PostgreSQL with persistent volumes
3. Deploy Redis cluster
4. Configure S3 access (AWS or MinIO)
5. Deploy backend with horizontal pod autoscaling
6. Deploy frontend with CDN
7. Configure ingress and TLS

Environment Variables

Production-specific variables:

```
# Database
POSTGRES_HOST=<your-postgres-host>
POSTGRES_PASSWORD=<strong-password>

# S3 Storage
S3_ENDPOINT_URL=https://s3.amazonaws.com
S3_ACCESS_KEY_ID=<aws-access-key>
S3_SECRET_ACCESS_KEY=<aws-secret-key>
S3_BUCKET_NAME=<production-bucket>

# Application
ENVIRONMENT=production
DEBUG=False
LOG_LEVEL=INFO
BACKEND_CORS_ORIGINS=https://your-domain.com

# Frontend
NEXT_PUBLIC_API_URL=https://api.your-domain.com/api/v1
```

Maintenance

Backup

Database:

```
docker-compose exec postgres pg_dump -U cosmology greds_library > backup.sql
```

FAISS/Whoosh Indexes:

```
docker-compose exec backend tar -czf /tmp/indexes.tar.gz /app/data
docker cp greds-backend:/tmp/indexes.tar.gz ./indexes-backup.tar.gz
```

S3 Audit Logs:

```
aws s3 sync s3://greds-audit-logs/ ./audit-logs-backup/
```

Updates

```
# Pull latest code
git pull origin main

# Rebuild containers
docker-compose build

# Restart services
docker-compose up -d

# Run new migrations
docker-compose exec backend alembic upgrade head
```

Monitoring

Monitor these metrics:

- API response times

- Database connection pool
- Redis queue depth
- S3 storage usage
- FAISS index size
- Error rates

Logs

```
# View all logs
docker-compose logs

# Follow specific service
docker-compose logs -f backend

# Export logs
docker-compose logs > system-logs.txt
```

Troubleshooting

Port Conflicts

If ports are already in use:

```
# Edit .env to change ports
BACKEND_PORT=8001
FRONTEND_PORT=3001
POSTGRES_PORT=5433
```

Memory Issues

If services crash due to memory:

```
# Increase Docker memory limit
# Docker Desktop > Settings > Resources > Memory

# Or reduce worker count in .env
```

Database Connection Issues

```
# Check PostgreSQL logs
docker-compose logs postgres

# Verify connection
docker-compose exec backend python -c "from app.db.session import engine; en-
gine.connect()"
```

Security Checklist

- [ ] Change default passwords
- [ ] Use strong JWT secret
- [ ] Enable S3 object locking
- [ ] Configure CORS appropriately

- [ ] Use HTTPS in production
- [ ] Implement rate limiting
- [ ] Enable audit logging
- [ ] Regular security updates
- [ ] Backup encryption
- [ ] Access control lists

Performance Tuning

Backend

- Adjust PostgreSQL connection pool size
- Configure Redis memory limits
- Optimize FAISS index parameters
- Enable result caching

Frontend

- Enable Next.js static optimization
- Configure CDN for static assets
- Implement API response caching
- Optimize bundle size

Database

- Create appropriate indexes
- Configure autovacuum
- Monitor slow queries
- Optimize connection pooling