Shell AI ESG Contract Analyzer - Complete Deployment Guide

Table of Contents

Prerequisites

Required Tools

```bash
# Install required CLI tools
brew install kubectl helm aws-cli terraform docker node@18

# Verify installations
kubectl version --client
helm version
aws --version
terraform --version
docker --version
node --version
```

AWS Account Setup

1. **Create IAM User for Deployment**

```bash
aws iam create-user --user-name esg-deployer
aws iam attach-user-policy --user-name esg-deployer --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
aws iam create-access-key --user-name esg-deployer
```

2. **Configure AWS CLI**

```bash
aws configure --profile esg-deployment
# Enter Access Key ID, Secret Access Key, Region (us-east-1), Output format (json)
```

Required Secrets

Create the following secrets in GitHub Repository Settings:

```yaml
# GitHub Secrets Required
AWS_ACCESS_KEY_ID: <from-iam-user>
AWS_SECRET_ACCESS_KEY: <from-iam-user>
AWS_REGION: us-east-1
OPENAI_API_KEY: <from-openai-dashboard>
OPENAI_API_KEY_TEST: <test-api-key>
JWT_SECRET: <generate-with-openssl>
DB_PASSWORD: <strong-password>
REDIS_PASSWORD: <strong-password>
GOOGLE_CLIENT_ID: <from-google-cloud>
GOOGLE_CLIENT_SECRET: <from-google-cloud>
MICROSOFT_CLIENT_ID: <from-azure-ad>
MICROSOFT_CLIENT_SECRET: <from-azure-ad>
SLACK_WEBHOOK: <from-slack-app>
CODECOV_TOKEN: <from-codecov>
SONAR_TOKEN: <from-sonarcloud>
GITGUARDIAN_API_KEY: <from-gitguardian>
```

Infrastructure Setup

1. Terraform Infrastructure

```bash
```

```
cd infrastructure/terraform

# Initialize Terraform
terraform init

# Plan infrastructure
terraform plan -out=tfplan

# Apply infrastructure
terraform apply tfplan
```

2. EKS Cluster Configuration

```
bash

# Update kubeconfig
aws eks update-kubeconfig --name shell-esg-cluster-prod --region us-east-1

# Verify cluster access
kubectl cluster-info
kubectl get nodes
```

3. Install Cluster Components

```
bash

# Install NGINX Ingress Controller
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm install ingress-nginx ingress-nginx/ingress-nginx \
  --namespace ingress-nginx \
  --create-namespace \
  --set controller.service.type=LoadBalancer

# Install Cert Manager for SSL
helm repo add jetstack https://charts.jetstack.io
helm install cert-manager jetstack/cert-manager \
  --namespace cert-manager \
  --create-namespace \
  --set installCRDs=true

# Install Prometheus & Grafana
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install kube-prometheus-stack prometheus-community/kube-prometheus-stack \
  --namespace monitoring \
  --create-namespace
```

4. Configure SSL Certificates

yaml

```yaml
# cert-issuer.yaml
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: devops@shell.com
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx
```

bash

```bash
kubectl apply -f cert-issuer.yaml
```

Development Environment

1. Local Development Setup

bash

```
# Clone repository
git clone https://github.com/shell/esg-contract-analyzer.git
cd esg-contract-analyzer

# Backend setup
cd backend
npm install
cp .env.example .env.local
# Edit .env.local with local values

# Frontend setup
cd ../frontend
npm install
cp .env.example .env.local
# Edit .env.local with local values

# Start services with Docker Compose
cd ..
docker-compose up -d
```

2. Database Migrations

```
bash

cd backend

# Run migrations
npm run migration:run

# Seed development data
npm run seed:dev
```

3. Running Tests Locally

```
bash

# Backend tests
cd backend
npm run test
npm run test:e2e

# Frontend tests
cd ../frontend
npm run test
npm run test:e2e
```

CI/CD Pipeline

1. Branch Strategy

```
main        → Production environment
staging     → Staging environment
develop     → Development environment
feature/*   → Feature branches (PR to develop)
hotfix/*    → Hotfix branches (PR to main)
```

2. Automated Deployment Flow

```
mermaid

graph TD
    A[Push to Branch] --> B{Which Branch?}
    B -->|feature/*| C[Run Tests]
    B -->|develop| D[Run Tests + Deploy to Dev]
    B -->|staging| E[Run Tests + Deploy to Staging]
    B -->|main| F[Run Tests + Deploy to Prod]
    C --> G[Create PR]
    D --> H[Dev Environment]
    E --> I[Staging Environment]
    F --> J[Production Environment]
```

3. Manual Deployment

```
bash

# Deploy specific version to staging
helm upgrade --install shell-esg-analyzer ./helm/shell-esg-analyzer \
  --namespace esg-staging \
  --values ./helm/shell-esg-analyzer/values.staging.yaml \
  --set image.tag=v1.2.3

# Deploy to production
helm upgrade --install shell-esg-analyzer ./helm/shell-esg-analyzer \
  --namespace esg-prod \
  --values ./helm/shell-esg-analyzer/values.prod.yaml \
  --set image.tag=v1.2.3
```

Deployment Environments

Environment Configuration

Environment	URL	Namespace	Resources
Development	https://dev.esg-analyzer.shell.com	esg-dev	2 replicas, 1GB RAM
Staging	https://staging.esg-analyzer.shell.com	esg-staging	3 replicas, 2GB RAM
Production	https://esg-analyzer.shell.com	esg-prod	5 replicas, 4GB RAM

Environment Variables

```yaml
# Common across all environments
NODE_ENV: <environment>
LOG_LEVEL: <info|debug|error>
CORS_ORIGIN: <frontend-url>

# Environment-specific
DATABASE_URL: postgresql://<user>:<pass>@<host>:<port>/<db>
REDIS_URL: redis://:<pass>@<host>:<port>
API_URL: https://api.<environment>.esg-analyzer.shell.com
```

Production Deployment

1. Pre-deployment Checklist

- [ ] All tests passing in CI/CD
- [ ] Security scan completed
- [ ] Database migrations reviewed
- [ ] Performance testing completed
- [ ] Rollback plan documented
- [ ] Team notified in Slack
- [ ] Maintenance window scheduled

2. Deployment Steps

```bash
```

```bash
# 1. Create release tag
git tag -a v1.2.3 -m "Release v1.2.3: ESG scoring improvements"
git push origin v1.2.3

# 2. Trigger deployment workflow
gh workflow run deploy.yml -f environment=production -f version=v1.2.3

# 3. Monitor deployment
kubectl get pods -n esg-prod -w
kubectl logs -f deployment/shell-esg-analyzer-backend -n esg-prod

# 4. Verify health checks
curl https://api.esg-analyzer.shell.com/health
curl https://esg-analyzer.shell.com
```

3. Post-deployment Verification

```bash
# Run smoke tests
npm run test:smoke -- --endpoint=https://esg-analyzer.shell.com

# Check metrics
kubectl top pods -n esg-prod
kubectl get hpa -n esg-prod

# Verify database
kubectl exec -it postgresql-0 -n esg-prod -- psql -U esguser -d esgdb -c "SELECT version();"
```

Monitoring and Maintenance

1. Application Monitoring

```bash
# Access Grafana dashboard
kubectl port-forward -n monitoring svc/kube-prometheus-stack-grafana 3000:80
# Open http://localhost:3000 (admin/prom-operator)

# Check application logs
kubectl logs -f deployment/shell-esg-analyzer-backend -n esg-prod
kubectl logs -f deployment/shell-esg-analyzer-frontend -n esg-prod
```

2. Database Maintenance

```bash
```

```
# Backup database
kubectl exec -n esg-prod postgresql-0 -- pg_dump -U esguser esgdb > backup_$(date +%Y%m%d).sql

# Analyze query performance
kubectl exec -it postgresql-0 -n esg-prod -- psql -U esguser -d esgdb
\x
SELECT * FROM pg_stat_statements ORDER BY total_time DESC LIMIT 10;
```

3. Regular Maintenance Tasks

yaml

```yaml
# cronjobs.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: database-backup
  namespace: esg-prod
spec:
  schedule: "0 2 * * *"  # Daily at 2 AM
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: backup
            image: postgres:15
            command:
            - /bin/bash
            - -c
            - |
              pg_dump -h postgresql -U esguser esgdb | \
              aws s3 cp - s3://shell-esg-backups/db/$(date +%Y%m%d).sql
```

Disaster Recovery

1. Backup Strategy

bash

```
# Automated backups
- Database: Daily snapshots to S3
- Files: Hourly sync to S3
- Configuration: Git repository
- Secrets: AWS Secrets Manager

# Manual backup
./scripts/backup-all.sh production
```

2. Recovery Procedures

```bash
# Database recovery
aws s3 cp s3://shell-esg-backups/db/20240315.sql ./
kubectl exec -i postgresql-0 -n esg-prod -- psql -U esguser esgdb < 20240315.sql

# Full cluster recovery
terraform apply -auto-approve
kubectl apply -k ./k8s/overlays/production
helm upgrade --install shell-esg-analyzer ./helm/shell-esg-analyzer -f values.prod.yaml
```

3. Rollback Procedure

```bash
# Quick rollback to previous version
helm rollback shell-esg-analyzer -n esg-prod

# Rollback with specific revision
helm history shell-esg-analyzer -n esg-prod
helm rollback shell-esg-analyzer 3 -n esg-prod

# Database migration rollback
cd backend
npm run migration:revert
```

Security Checklist

Pre-deployment Security

- [ ] All dependencies updated (npm audit fix)
- [ ] No secrets in code (GitGuardian scan)
- [ ] Docker images scanned (Trivy)
- [ ] OWASP Top 10 compliance verified

- [ ] SSL certificates valid
- [ ] WAF rules updated
- [ ] Rate limiting configured
- [ ] CORS properly configured

Runtime Security

```bash
# Enable Pod Security Policies
kubectl apply -f security/pod-security-policy.yaml

# Network Policies
kubectl apply -f security/network-policies.yaml

# Secret rotation
kubectl create secret generic app-secrets \
  --from-literal=jwt-secret=$(openssl rand -base64 32) \
  --dry-run=client -o yaml | kubectl apply -f -
```

Troubleshooting

Common Issues

1. Pod CrashLoopBackOff

```bash
# Check logs
kubectl logs <pod-name> -n esg-prod --previous

# Check events
kubectl describe pod <pod-name> -n esg-prod

# Common fixes:
# - Check environment variables
# - Verify database connectivity
# - Check resource limits
```

2. Database Connection Issues

```bash
```

```
# Test connection from pod
kubectl exec -it deployment/shell-esg-analyzer-backend -n esg-prod -- \
  nc -zv postgresql 5432

# Check database pod
kubectl logs postgresql-0 -n esg-prod
```

3. High Memory Usage

```
bash

# Check current usage
kubectl top pods -n esg-prod

# Scale horizontally
kubectl scale deployment shell-esg-analyzer-backend --replicas=10 -n esg-prod

# Or update resources
kubectl set resources deployment shell-esg-analyzer-backend \
  --limits=memory=4Gi --requests=memory=2Gi -n esg-prod
```

Emergency Contacts

- **DevOps Lead**: devops-lead@shell.com (+1-xxx-xxx-xxxx)
- **Security Team**: security@shell.com
- **Database Admin**: dba@shell.com
- **On-call Engineer**: Check PagerDuty

Useful Commands Reference

```
bash
```

```
# Get all resources in namespace
kubectl get all -n esg-prod

# Forward port for debugging
kubectl port-forward svc/shell-esg-analyzer-backend 8080:3000 -n esg-prod

# Execute commands in container
kubectl exec -it deployment/shell-esg-analyzer-backend -n esg-prod -- /bin/bash

# Check resource usage
kubectl top nodes
kubectl top pods -n esg-prod

# View recent events
kubectl get events -n esg-prod --sort-by='.lastTimestamp'
```

Conclusion

This deployment guide covers the complete lifecycle of deploying the Shell AI ESG Contract Analyzer. Always ensure to:

1. Follow the deployment checklist
2. Monitor the application post-deployment
3. Keep backups current
4. Maintain security best practices
5. Document any deviations or custom configurations

For additional support, contact the DevOps team or refer to the internal wiki.