# Guia Completo de Deployment em Produção - PrediTest AI

## 📋 Índice

---

## 🎯 Visão Geral

Este guia fornece instruções passo a passo para fazer deploy da aplicação PrediTest AI em um ambiente de produção na nuvem com alta disponibilidade, escalabilidade e segurança.

### Requisitos de Produção

- **Uptime**: ⩾99.9%

- **Latência P95**: <200ms
- **Throughput**: 1000 req/s
- **Escalabilidade**: Auto-scaling horizontal
- **Backup**: Diário com retenção de 30 dias
- **Segurança**: HTTPS/TLS, OAuth 2.0, JWT

---

## 🏗️ Arquitetura de Produção

```
┌─────────────────────────────────────────────────────────────┐
│                        CDN GLOBAL                            │
│                  (CloudFlare/CloudFront)                     │
└─────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────┐
│                      LOAD BALANCER                           │
│               (Application Load Balancer)                    │
└─────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────┐
│                   KUBERNETES CLUSTER                         │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐      │
│  │   Frontend   │  │   Backend    │  │   Backend    │      │
│  │   Pod 1      │  │   Pod 1      │  │   Pod 2      │      │
│  └──────────────┘  └──────────────┘  └──────────────┘      │
│                                                              │
│  ┌──────────────────────────────────────────────────────┐  │
│  │           Ingress Controller (Nginx)                 │  │
│  └──────────────────────────────────────────────────────┘  │
└─────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────┐
│                       DATA LAYER                             │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐      │
│  │ PostgreSQL   │  │    Redis     │  │  S3/Storage  │      │
│  │  (Primary)   │  │  (Cluster)   │  │  (Backups)   │      │
│  └──────────────┘  └──────────────┘  └──────────────┘      │
└─────────────────────────────────────────────────────────────┘
                              ↓
┌─────────────────────────────────────────────────────────────┐
│                  MONITORING & LOGGING                        │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐      │
│  │ Prometheus   │  │   Grafana    │  │  ELK Stack   │      │
│  └──────────────┘  └──────────────┘  └──────────────┘      │
└─────────────────────────────────────────────────────────────┘
```

---

## ✅ Pré-requisitos

### Ferramentas Necessárias

```
 # Instalar ferramentas CLI
brew install awscli          # AWS CLI
brew install gcloud          # Google Cloud SDK
brew install az              # Azure CLI
brew install kubectl         # Kubernetes
brew install helm            # Helm (Kubernetes package manager)
brew install docker          # Docker
brew install terraform       # Infrastructure as Code (opcional)
```

### Contas de Cloud

- ✅ Conta AWS com permissões de IAM
- ✅ Conta Google Cloud com projeto criado
- ✅ Conta Azure com subscription ativa
- ✅ Conta Docker Hub para armazenar imagens

### Domínio

- ✅ Domínio registrado (ex: preditest-ai.nestle.com.br)
- ✅ Certificado SSL/TLS (Let's Encrypt ou AWS Certificate Manager)

---

## ☁️ Opções de Cloud

### AWS (Recomendado)

- **Serviços**: ECS/EKS, RDS, ElastiCache, S3, CloudFront
- **Custo Estimado**: $2,000-3,000/mês
- **Vantagens**: Maior ecossistema, melhor suporte
- **Documentação**: https://aws.amazon.com

### Google Cloud

- **Serviços**: GKE, Cloud SQL, Memorystore, Cloud Storage, Cloud CDN
- **Custo Estimado**: $1,800-2,800/mês
- **Vantagens**: Melhor ML/IA, preços competitivos
- **Documentação**: https://cloud.google.com

### Azure

- **Serviços**: AKS, Azure Database, Azure Cache, Blob Storage, CDN
- **Custo Estimado**: $1,900-2,900/mês
- **Vantagens**: Integração com Microsoft, enterprise-friendly
- **Documentação**: https://azure.microsoft.com

---

# 🚀 Deployment no AWS

## Passo 1: Preparar Imagem Docker

```
# 1.1 Criar Dockerfile
cat > Dockerfile << 'EOF'
FROM node:22-alpine

WORKDIR /app

# Instalar pnpm
RUN npm install -g pnpm

# Copiar arquivos
COPY package.json pnpm-lock.yaml ./
RUN pnpm install --frozen-lockfile

COPY . .

# Build
RUN pnpm build

# Expor porta
EXPOSE 3000

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD node -e "require('http').get('http://localhost:3000/health', (r) => {if
(r.statusCode !== 200) throw new Error(r.statusCode)})"

# Iniciar
CMD ["pnpm", "start"]
EOF

# 1.2 Build da imagem
docker build -t preditest-ai:latest .

# 1.3 Tag para AWS ECR
docker tag preditest-ai:latest 123456789.dkr.ecr.us-east-
1.amazonaws.com/preditest-ai:latest

# 1.4 Push para ECR
aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 123456789.dkr.ecr.us-east-1.amazonaws.com
docker push 123456789.dkr.ecr.us-east-1.amazonaws.com/preditest-ai:latest
```

## Passo 2: Criar Banco de Dados RDS

```
 # 2.1 Criar instância PostgreSQL RDS
aws rds create-db-instance \
  --db-instance-identifier preditest-ai-db \
  --db-instance-class db.t3.medium \
  --engine postgres \
  --engine-version 15.3 \
  --master-username admin \
  --master-user-password 'YourSecurePassword123!' \
  --allocated-storage 100 \
  --storage-type gp3 \
  --multi-az \
  --backup-retention-period 30 \
  --publicly-accessible false \
  --vpc-security-group-ids sg-xxxxx

# 2.2 Aguardar criação (5-10 minutos)
aws rds describe-db-instances --db-instance-identifier preditest-ai-db

# 2.3 Obter endpoint
aws rds describe-db-instances \
  --db-instance-identifier preditest-ai-db \
  --query 'DBInstances[0].Endpoint.Address'
```

## Passo 3: Criar Cache Redis

```
 # 3.1 Criar cluster ElastiCache Redis
aws elasticache create-cache-cluster \
  --cache-cluster-id preditest-ai-redis \
  --cache-node-type cache.t3.micro \
  --engine redis \
  --engine-version 7.0 \
  --num-cache-nodes 1 \
  --security-group-ids sg-xxxxx

# 3.2 Para produção, usar replication group
aws elasticache create-replication-group \
  --replication-group-description "PrediTest AI Redis" \
  --replication-group-id preditest-ai-redis \
  --engine redis \
  --cache-node-type cache.t3.small \
  --num-cache-clusters 3 \
  --automatic-failover-enabled \
  --multi-az-enabled
```

## Passo 4: Criar Cluster EKS

```
# 4.1 Criar cluster Kubernetes
eksctl create cluster \
  --name preditest-ai \
  --version 1.28 \
  --region us-east-1 \
  --nodegroup-name standard-nodes \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 2 \
  --nodes-max 10 \
  --enable-ssm \
  --with-oidc

# 4.2 Configurar kubectl
aws eks update-kubeconfig \
  --region us-east-1 \
  --name preditest-ai

# 4.3 Verificar cluster
kubectl get nodes
```

## Passo 5: Configurar Secrets no Kubernetes

```
# 5.1 Criar namespace
kubectl create namespace preditest-ai

# 5.2 Criar secrets
kubectl create secret generic preditest-ai-secrets \
  --from-literal=DATABASE_URL='postgresql://admin:password@preditest-ai-
db.xxxxx.rds.amazonaws.com:5432/preditest_ai' \
  --from-literal=JWT_SECRET='your-secret-key-here' \
  --from-literal=VITE_APP_ID='your-app-id' \
  --from-literal=OAUTH_SERVER_URL='https://api.manus.im' \
  --from-literal=BUILT_IN_FORGE_API_KEY='your-api-key' \
  -n preditest-ai

# 5.3 Verificar secrets
kubectl get secrets -n preditest-ai
```

## Passo 6: Deploy no Kubernetes

```
# 6.1 Criar arquivo deployment.yaml
cat > k8s/deployment.yaml << 'EOF'
apiVersion: apps/v1
kind: Deployment
metadata:
  name: preditest-ai
  namespace: preditest-ai
spec:
  replicas: 3
  selector:
    matchLabels:
      app: preditest-ai
  template:
    metadata:
      labels:
        app: preditest-ai
    spec:
      containers:
      - name: preditest-ai
        image: 123456789.dkr.ecr.us-east-1.amazonaws.com/preditest-ai:latest
        ports:
        - containerPort: 3000
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: preditest-ai-secrets
              key: DATABASE_URL
        - name: JWT_SECRET
          valueFrom:
            secretKeyRef:
              name: preditest-ai-secrets
              key: JWT_SECRET
        resources:
          requests:
            cpu: 500m
            memory: 512Mi
          limits:
            cpu: 1000m
            memory: 1Gi
        livenessProbe:
          httpGet:
            path: /health
            port: 3000
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 3000
          initialDelaySeconds: 5
          periodSeconds: 5
EOF

# 6.2 Criar Service
cat > k8s/service.yaml << 'EOF'
apiVersion: v1
kind: Service
metadata:
  name: preditest-ai
  namespace: preditest-ai
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 3000
    protocol: TCP
  selector:
    app: preditest-ai
EOF

# 6.3 Criar Ingress
cat > k8s/ingress.yaml << 'EOF'
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: preditest-ai
  namespace: preditest-ai
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    nginx.ingress.kubernetes.io/ssl-redirect: "true"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - preditest-ai.nestle.com.br
    secretName: preditest-ai-tls
  rules:
  - host: preditest-ai.nestle.com.br
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: preditest-ai
            port:
              number: 80
EOF

# 6.4 Aplicar manifests
kubectl apply -f k8s/deployment.yaml
kubectl apply -f k8s/service.yaml
kubectl apply -f k8s/ingress.yaml

# 6.5 Verificar deployment
kubectl get pods -n preditest-ai
kubectl logs -f deployment/preditest-ai -n preditest-ai
```

## Passo 7: Configurar Auto-scaling

```
# 7.1 Criar Horizontal Pod Autoscaler
cat > k8s/hpa.yaml << 'EOF'
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: preditest-ai-hpa
  namespace: preditest-ai
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: preditest-ai
  minReplicas: 3
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
EOF

kubectl apply -f k8s/hpa.yaml

# 7.2 Verificar HPA
kubectl get hpa -n preditest-ai
```

## Passo 8: Configurar CDN (CloudFront)

```
# 8.1 Criar distribuição CloudFront
aws cloudfront create-distribution \
  --origin-domain-name preditest-ai.nestle.com.br \
  --default-root-object index.html \
  --viewer-protocol-policy redirect-to-https \
  --cache-behaviors '[
    {
      "PathPattern": "/api/*",
      "ViewerProtocolPolicy": "https-only",
      "CachePolicyId": "4135ea3d-c35d-46eb-81d7-reeSQe797d63"
    }
  ]'
```

---

# 🌐 Deployment no Google Cloud

## Passo 1: Preparar Projeto GCP

```
# 1.1 Configurar projeto
gcloud config set project preditest-ai-prod

# 1.2 Habilitar APIs necessárias
gcloud services enable \
  container.googleapis.com \
  sqladmin.googleapis.com \
  redis.googleapis.com \
  storage-api.googleapis.com
```

## Passo 2: Criar Cluster GKE

```
# 2.1 Criar cluster
gcloud container clusters create preditest-ai \
  --region us-central1 \
  --num-nodes 3 \
  --machine-type n1-standard-2 \
  --enable-autoscaling \
  --min-nodes 2 \
  --max-nodes 10 \
  --enable-autorepair \
  --enable-autoupgrade

# 2.2 Configurar kubectl
gcloud container clusters get-credentials preditest-ai --region us-central1
```

## Passo 3: Criar Cloud SQL (PostgreSQL)

```
# 3.1 Criar instância
gcloud sql instances create preditest-ai-db \
  --database-version POSTGRES_15 \
  --tier db-custom-2-8192 \
  --region us-central1 \
  --backup-start-time 02:00 \
  --enable-bin-log \
  --retained-backups-count 30

# 3.2 Criar banco de dados
gcloud sql databases create preditest_ai \
  --instance=preditest-ai-db

# 3.3 Criar usuário
gcloud sql users create preditest_user \
  --instance=preditest-ai-db \
  --password=YourSecurePassword123!
```

### Passo 4: Criar Memorystore (Redis)

```
# 4.1 Criar instância Redis
gcloud redis instances create preditest-ai-redis \
  --size=2 \
  --region=us-central1 \
  --redis-version=7.0 \
  --tier=standard
```

### Passo 5: Push da Imagem Docker

```
# 5.1 Configurar Docker
gcloud auth configure-docker gcr.io

# 5.2 Build e push
docker build -t gcr.io/preditest-ai-prod/preditest-ai:latest .
docker push gcr.io/preditest-ai-prod/preditest-ai:latest
```

### Passo 6: Deploy no GKE

```
# 6.1 Criar namespace
kubectl create namespace preditest-ai

# 6.2 Criar secrets
kubectl create secret generic preditest-ai-secrets \
  --from-literal=DATABASE_URL='postgresql://...' \
  --from-literal=JWT_SECRET='...' \
  -n preditest-ai

# 6.3 Aplicar manifests Kubernetes (mesmo do AWS)
kubectl apply -f k8s/ -n preditest-ai
```

---

## 🔷 Deployment no Azure

### Passo 1: Preparar Ambiente Azure

```bash
# 1.1 Fazer login
az login

# 1.2 Criar resource group
az group create \
  --name preditest-ai-rg \
  --location eastus

# 1.3 Criar container registry
az acr create \
  --resource-group preditest-ai-rg \
  --name preditestairegistry \
  --sku Basic
```

### Passo 2: Criar AKS Cluster

```bash
# 2.1 Criar cluster
az aks create \
  --resource-group preditest-ai-rg \
  --name preditest-ai-aks \
  --node-count 3 \
  --vm-set-type VirtualMachineScaleSets \
  --load-balancer-sku standard \
  --enable-managed-identity \
  --network-plugin azure

# 2.2 Configurar kubectl
az aks get-credentials \
  --resource-group preditest-ai-rg \
  --name preditest-ai-aks
```

### Passo 3: Criar Azure Database for PostgreSQL

```bash
# 3.1 Criar servidor
az postgres server create \
  --resource-group preditest-ai-rg \
  --name preditest-ai-db \
  --location eastus \
  --admin-user dbadmin \
  --admin-password YourSecurePassword123! \
  --sku-name B_Gen5_2 \
  --storage-size 51200 \
  --backup-retention 30

# 3.2 Criar banco de dados
az postgres db create \
  --resource-group preditest-ai-rg \
  --server-name preditest-ai-db \
  --name preditest_ai
```

### Passo 4: Criar Azure Cache for Redis

```
# 4.1 Criar cache
az redis create \
  --resource-group preditest-ai-rg \
  --name preditest-ai-redis \
  --location eastus \
  --sku Basic \
  --vm-size c0
```

### Passo 5: Push da Imagem

```
# 5.1 Login no ACR
az acr login --name preditestairegistry

# 5.2 Build e push
docker build -t preditestairegistry.azurecr.io/preditest-ai:latest .
docker push preditestairegistry.azurecr.io/preditest-ai:latest
```

---

## 🔄 Configuração de CI/CD

### GitHub Actions

```yaml
# .github/workflows/deploy.yml
name: Deploy to Production

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Setup Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '22'

    - name: Install dependencies
      run: npm install -g pnpm && pnpm install

    - name: Run tests
      run: pnpm test

    - name: Build
      run: pnpm build

    - name: Build Docker image
      run: docker build -t preditest-ai:${{ github.sha }} .

    - name: Push to ECR
      env:
        AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
        AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
      run: |
        aws ecr get-login-password --region us-east-1 | docker login --username
AWS --password-stdin 123456789.dkr.ecr.us-east-1.amazonaws.com
        docker tag preditest-ai:${{ github.sha }} 123456789.dkr.ecr.us-east-
1.amazonaws.com/preditest-ai:latest
        docker push 123456789.dkr.ecr.us-east-1.amazonaws.com/preditest-
ai:latest

    - name: Deploy to EKS
      env:
        KUBECONFIG: ${{ secrets.KUBECONFIG }}
      run: |
        kubectl set image deployment/preditest-ai preditest-
ai=123456789.dkr.ecr.us-east-1.amazonaws.com/preditest-ai:latest -n preditest-
ai
        kubectl rollout status deployment/preditest-ai -n preditest-ai
```

# 📊 Monitoramento e Observabilidade

## Prometheus + Grafana

```
# Instalar Prometheus
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm install prometheus prometheus-community/kube-prometheus-stack \
  -n preditest-ai

# Instalar Grafana
helm repo add grafana https://grafana.github.io/helm-charts
helm install grafana grafana/grafana -n preditest-ai
```

## ELK Stack (Elasticsearch, Logstash, Kibana)

```
# Instalar ELK
helm repo add elastic https://helm.elastic.co
helm install elasticsearch elastic/elasticsearch -n preditest-ai
helm install kibana elastic/kibana -n preditest-ai
```

## CloudWatch (AWS)

```
# Criar log group
aws logs create-log-group --log-group-name /preditest-ai/app

# Criar alarmes
aws cloudwatch put-metric-alarm \
  --alarm-name preditest-ai-high-cpu \
  --alarm-description "Alert when CPU is high" \
  --metric-name CPUUtilization \
  --namespace AWS/ECS \
  --statistic Average \
  --period 300 \
  --threshold 80 \
  --comparison-operator GreaterThanThreshold
```

## 🔐 Segurança em Produção

### Certificado SSL/TLS

```
# Instalar cert-manager
helm repo add jetstack https://charts.jetstack.io
helm install cert-manager jetstack/cert-manager -n cert-manager --create-
namespace

# Criar ClusterIssuer
cat > k8s/cert-issuer.yaml << 'EOF'
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-prod
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: admin@nestle.com.br
    privateKeySecretRef:
      name: letsencrypt-prod
    solvers:
    - http01:
        ingress:
          class: nginx
EOF

kubectl apply -f k8s/cert-issuer.yaml
```

## Network Policies

```
# Criar Network Policy
cat > k8s/network-policy.yaml << 'EOF'
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: preditest-ai-netpol
  namespace: preditest-ai
spec:
  podSelector:
    matchLabels:
      app: preditest-ai
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    ports:
    - protocol: TCP
      port: 3000
  egress:
  - to:
    - namespaceSelector: {}
    ports:
    - protocol: TCP
      port: 5432
    - protocol: TCP
      port: 6379
EOF

kubectl apply -f k8s/network-policy.yaml
```

## Secrets Management

```
# Usar AWS Secrets Manager
aws secretsmanager create-secret \
  --name preditest-ai/prod \
  --secret-string file://secrets.json

# Ou usar HashiCorp Vault
helm repo add hashicorp https://helm.releases.hashicorp.com
helm install vault hashicorp/vault -n preditest-ai
```

---

## 💾 Backup e Disaster Recovery

### Backup Automático

```
# AWS RDS - Backup automático (já configurado)
aws rds modify-db-instance \
  --db-instance-identifier preditest-ai-db \
  --backup-retention-period 30 \
  --preferred-backup-window "02:00-03:00"

# Backup manual
aws rds create-db-snapshot \
  --db-instance-identifier preditest-ai-db \
  --db-snapshot-identifier preditest-ai-backup-$(date +%Y%m%d)
```

### Disaster Recovery

```
# Criar read replica (para failover)
aws rds create-db-instance-read-replica \
  --db-instance-identifier preditest-ai-db-replica \
  --source-db-instance-identifier preditest-ai-db \
  --availability-zone us-east-1b

# Promover read replica em caso de falha
aws rds promote-read-replica \
  --db-instance-identifier preditest-ai-db-replica
```

### Velero (Backup Kubernetes)

```
# Instalar Velero
wget https://github.com/vmware-tanzu/velero/releases/download/v1.11.0/velero-
v1.11.0-linux-amd64.tar.gz
tar -xvf velero-v1.11.0-linux-amd64.tar.gz
sudo mv velero-v1.11.0-linux-amd64/velero /usr/local/bin/

# Configurar backup
velero install \
  --provider aws \
  --bucket preditest-ai-backups \
  --secret-file ./credentials-velero

# Criar schedule de backup
velero schedule create preditest-ai-daily \
  --schedule="0 2 * * *" \
  --include-namespaces preditest-ai
```

---

## 🔧 Troubleshooting

### Problemas Comuns

#### 1. Pod não inicia

```
# Verificar logs
kubectl logs pod/preditest-ai-xxxxx -n preditest-ai

# Descrever pod
kubectl describe pod/preditest-ai-xxxxx -n preditest-ai

# Verificar eventos
kubectl get events -n preditest-ai
```

#### 2. Conexão com banco de dados falha

```
# Testar conectividade
kubectl run -it --rm debug --image=busybox --restart=Never -- \
  nc -zv preditest-ai-db.xxxxx.rds.amazonaws.com 5432

# Verificar secrets
kubectl get secret preditest-ai-secrets -n preditest-ai -o yaml
```

#### 3. Alto uso de memória

```
# Verificar uso de recursos
kubectl top pods -n preditest-ai

# Aumentar limites
kubectl set resources deployment preditest-ai \
  --limits=memory=2Gi,cpu=2 \
  --requests=memory=1Gi,cpu=1 \
  -n preditest-ai
```

#### 4. Aplicação lenta

```
# Verificar latência
kubectl exec -it pod/preditest-ai-xxxxx -n preditest-ai -- \
  curl -w "@curl-format.txt" -o /dev/null -s http://localhost:3000

# Verificar cache Redis
redis-cli -h preditest-ai-redis.xxxxx.cache.amazonaws.com INFO stats
```

## 📋 Checklist de Deployment

### Pré-Deployment

- [ ] Testes unitários passando
- [ ] Testes de integração passando
- [ ] Build Docker funcionando
- [ ] Variáveis de ambiente configuradas
- [ ] Certificado SSL/TLS válido
- [ ] Backup do banco de dados realizado

### Deployment

- [ ] Imagem Docker enviada para registry
- [ ] Secrets criados no Kubernetes
- [ ] Deployment aplicado
- [ ] Pods iniciando corretamente
- [ ] Serviço acessível
- [ ] Ingress funcionando

### Pós-Deployment

- [ ] Monitoramento ativo
- [ ] Alertas configurados
- [ ] Logs sendo coletados
- [ ] Backup automático ativo
- [ ] Testes de smoke passando
- [ ] Performance dentro dos limites

---

## 📞 Suporte e Escalação

**Em caso de problemas:**

1. Verificar logs: `kubectl logs -f deployment/preditest-ai -n preditest-ai`
2. Verificar eventos: `kubectl get events -n preditest-ai`
3. Verificar métricas: Acessar Grafana/CloudWatch
4. Contatar suporte: innovation@nestle.com.br

---

## 📚 Referências

- AWS EKS Documentation
- Google GKE Documentation
- Azure AKS Documentation
- Kubernetes Official Docs
- Helm Documentation

---

**Versão**: 1.0.0 | **Data**: Outubro 2025 | **Status**: Pronto para Produção

Desenvolvido com ❤️ para a Nestlé Brasil