🚀 Production Deployment Guide

Complete Infrastructure Setup & CI/CD Pipeline

📋 Overview

This guide covers:

1. **Infrastructure Setup** - AWS/GCP/DigitalOcean deployment
2. **Docker & Containerization** - Optimize images
3. **Kubernetes Deployment** - Scale to production
4. **CI/CD Pipeline** - GitHub Actions / GitLab CI
5. **SSL/TLS & Security** - HTTPS, encryption, security groups
6. **Database Backup & Recovery** - PostgreSQL backup strategy
7. **Monitoring & Alerting** - Production monitoring setup
8. **Load Balancing & Auto-Scaling** - Handle traffic spikes
9. **Disaster Recovery** - Failover & recovery procedures
10. **Cost Optimization** - Reduce cloud spending

1️⃣ Infrastructure Setup

🏗️ AWS Architecture (Recommended)

File: `infrastructure/terraform/main.tf`

```
# AWS Provider
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~&gt; 5.0"
    }
  }
}

provider "aws" {
  region = var.aws_region
}

# VPC Setup
resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
```

```
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    Name = "construction-ai-vpc"
  }
}

# Public Subnets
resource "aws_subnet" "public_1" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = "10.0.1.0/24"
  availability_zone       = "${var.aws_region}a"
  map_public_ip_on_launch = true

  tags = {
    Name = "public-subnet-1"
  }
}

resource "aws_subnet" "public_2" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = "10.0.2.0/24"
  availability_zone       = "${var.aws_region}b"
  map_public_ip_on_launch = true

  tags = {
    Name = "public-subnet-2"
  }
}

# Private Subnets (for databases)
resource "aws_subnet" "private_1" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.10.0/24"
  availability_zone = "${var.aws_region}a"

  tags = {
    Name = "private-subnet-1"
  }
}

resource "aws_subnet" "private_2" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.11.0/24"
  availability_zone = "${var.aws_region}b"

  tags = {
    Name = "private-subnet-2"
  }
}

# Internet Gateway
resource "aws_internet_gateway" "main" {
  vpc_id = aws_vpc.main.id
```

```
  tags = {
    Name = "construction-ai-igw"
  }
}

# Route Table for Public Subnets
resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block      = "0.0.0.0/0"
    gateway_id      = aws_internet_gateway.main.id
  }

  tags = {
    Name = "public-route-table"
  }
}

resource "aws_route_table_association" "public_1" {
  subnet_id      = aws_subnet.public_1.id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table_association" "public_2" {
  subnet_id      = aws_subnet.public_2.id
  route_table_id = aws_route_table.public.id
}

# Security Group for ALB
resource "aws_security_group" "alb" {
  name   = "construction-ai-alb-sg"
  vpc_id = aws_vpc.main.id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
```

```
    Name = "alb-security-group"
  }
}

# Security Group for ECS
resource "aws_security_group" "ecs" {
  name   = "construction-ai-ecs-sg"
  vpc_id = aws_vpc.main.id

  ingress {
    from_port       = 0
    to_port         = 65535
    protocol        = "tcp"
    security_groups = [aws_security_group.alb.id]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "ecs-security-group"
  }
}

# RDS PostgreSQL
resource "aws_rds_cluster" "postgres" {
  cluster_identifier      = "construction-ai-db"
  engine                  = "aurora-postgresql"
  engine_version          = "15.3"
  database_name           = "construction_ai"
  master_username         = var.db_username
  master_password         = var.db_password

  db_subnet_group_name    = aws_db_subnet_group.main.name
  vpc_security_group_ids  = [aws_security_group.db.id]

  backup_retention_period = 30
  enabled_cloudwatch_logs_exports = ["postgresql"]

  skip_final_snapshot     = false
  final_snapshot_identifier = "construction-ai-db-final-snapshot-${formatdate("YYYY-MM-DD

  tags = {
    Name = "construction-ai-database"
  }
}

# ECS Cluster
resource "aws_ecs_cluster" "main" {
  name = "construction-ai-cluster"

  setting {
```

```
    name  = "containerInsights"
    value = "enabled"
  }

  tags = {
    Name = "construction-ai-ecs-cluster"
  }
}

# ECR Repository for Docker images
resource "aws_ecr_repository" "api" {
  name                 = "construction-ai/api"
  image_tag_mutability = "IMMUTABLE"

  image_scanning_configuration {
    scan_on_push = true
  }

  tags = {
    Name = "api-repository"
  }
}

resource "aws_ecr_repository" "ui" {
  name                 = "construction-ai/ui"
  image_tag_mutability = "IMMUTABLE"

  image_scanning_configuration {
    scan_on_push = true
  }

  tags = {
    Name = "ui-repository"
  }
}

# ALB
resource "aws_lb" "main" {
  name               = "construction-ai-alb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb.id]
  subnets            = [aws_subnet.public_1.id, aws_subnet.public_2.id]

  enable_deletion_protection = false

  tags = {
    Name = "main-alb"
  }
}

# S3 for file uploads
resource "aws_s3_bucket" "uploads" {
  bucket = "construction-ai-uploads-${data.aws_caller_identity.current.account_id}"

  tags = {
```

```
    Name = "uploads-bucket"
  }
}

resource "aws_s3_bucket_versioning" "uploads" {
  bucket = aws_s3_bucket.uploads.id

  versioning_configuration {
    status = "Enabled"
  }
}

# S3 for archival (Glacier)
resource "aws_s3_bucket" "archive" {
  bucket = "construction-ai-archive-${data.aws_caller_identity.current.account_id}"

  tags = {
    Name = "archive-bucket"
  }
}

resource "aws_s3_bucket_lifecycle_configuration" "archive" {
  bucket = aws_s3_bucket.archive.id

  rule {
    id     = "archive-to-glacier"
    status = "Enabled"

    transition {
      days          = 30
      storage_class = "GLACIER"
    }
  }
}

# CloudFront CDN
resource "aws_cloudfront_distribution" "main" {
  origin {
    domain_name = aws_lb.main.dns_name
    origin_id   = "alb"

    custom_origin_config {
      http_port              = 80
      https_port             = 443
      origin_protocol_policy = "https-only"
      origin_ssl_protocols   = ["TLSv1.2"]
    }
  }

  enabled             = true
  is_ipv6_enabled     = true
  default_root_object = "index.html"

  default_cache_behavior {
    allowed_methods  = ["DELETE", "GET", "HEAD", "OPTIONS", "PATCH", "POST", "PUT"]
    cached_methods   = ["GET", "HEAD"]
```

```
    target_origin_id = "alb"

    forwarded_values {
      query_string = true
      cookies {
        forward = "all"
      }
      headers = ["*"]
    }

    viewer_protocol_policy = "redirect-to-https"
    min_ttl                = 0
    default_ttl            = 3600
    max_ttl                = 86400
  }

  restrictions {
    geo_restriction {
      restriction_type = "none"
    }
  }

  viewer_certificate {
    cloudfront_default_certificate = true
  }

  tags = {
    Name = "main-cdn"
  }
}

# Outputs
output "alb_dns_name" {
  value = aws_lb.main.dns_name
}

output "cloudfront_domain_name" {
  value = aws_cloudfront_distribution.main.domain_name
}

output "rds_endpoint" {
  value = aws_rds_cluster.postgres.endpoint
}
```

2 Docker Optimization

Optimized Dockerfiles

File: `python-services/api/Dockerfile.prod`

```
# Build stage
FROM python:3.11-slim as builder
```

```
WORKDIR /app

# Install build dependencies
RUN apt-get update &amp;&amp; apt-get install -y --no-install-recommends \
    gcc \
    postgresql-client \
    &amp;&amp; rm -rf /var/lib/apt/lists/*

# Copy requirements
COPY requirements.txt .

# Install Python dependencies
RUN pip install --user --no-cache-dir -r requirements.txt

# Runtime stage
FROM python:3.11-slim

WORKDIR /app

# Install runtime dependencies only
RUN apt-get update &amp;&amp; apt-get install -y --no-install-recommends \
    postgresql-client \
    &amp;&amp; rm -rf /var/lib/apt/lists/*

# Copy Python dependencies from builder
COPY --from=builder /root/.local /root/.local

# Set PATH
ENV PATH=/root/.local/bin:$PATH
ENV PYTHONUNBUFFERED=1

# Copy application
COPY . .

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=40s --retries=3 \
    CMD python -c "import requests; requests.get('http://localhost:8000/health')"

# Run with gunicorn
CMD ["gunicorn", "-w", "4", "-k", "uvicorn.workers.UvicornWorker", "--bind", "0.0.0.0:800
```

File: `web-react/Dockerfile.prod`

```
# Build stage
FROM node:18-alpine as builder

WORKDIR /app

COPY package*.json ./
RUN npm ci

COPY . .
RUN npm run build

# Runtime stage
```

```
FROM node:18-alpine

WORKDIR /app

RUN npm install -g serve

COPY --from=builder /app/build ./build

HEALTHCHECK --interval=30s --timeout=10s --retries=3 \
    CMD wget --quiet --tries=1 --spider http://localhost:3000/health || exit 1

EXPOSE 3000

CMD ["serve", "-s", "build", "-l", "3000"]
```

3 Kubernetes Deployment

☸ Kubernetes Configuration

File: `k8s/namespace.yaml`

```
apiVersion: v1
kind: Namespace
metadata:
  name: construction-ai
  labels:
    name: construction-ai
```

File: `k8s/api-deployment.yaml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api
  namespace: construction-ai
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: api
  template:
    metadata:
      labels:
        app: api
    spec:
      containers:
```

```
      - name: api
        image: ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/construction-ai/api:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 8000
          name: http
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: connection-string
        - name: REDIS_URL
          valueFrom:
            configMapKeyRef:
              name: redis-config
              key: url
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /health/live
            port: 8000
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
          failureThreshold: 3
        readinessProbe:
          httpGet:
            path: /health/ready
            port: 8000
          initialDelaySeconds: 20
          periodSeconds: 5
          timeoutSeconds: 5
          failureThreshold: 3
      affinity:
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - api
              topologyKey: kubernetes.io/hostname

---
apiVersion: v1
kind: Service
```

```
metadata:
  name: api
  namespace: construction-ai
spec:
  type: ClusterIP
  ports:
  - port: 8000
    targetPort: 8000
    protocol: TCP
  selector:
    app: api

---
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: api-hpa
  namespace: construction-ai
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: api
  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
```

File: `k8s/ingress.yaml`

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: construction-ai
  namespace: construction-ai
  annotations:
    cert-manager.io/cluster-issuer: "letsencrypt-prod"
    nginx.ingress.kubernetes.io/rate-limit: "100"
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - api.construction-ai.com
    - app.construction-ai.com
```

```
    secretName: construction-ai-tls
  rules:
  - host: api.construction-ai.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: api
            port:
              number: 8000
  - host: app.construction-ai.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: ui
            port:
              number: 3000
```

4 CI/CD Pipeline

GitHub Actions Pipeline

File: `.github/workflows/deploy.yml`

```
name: Build and Deploy

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

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}
  AWS_REGION: us-east-1

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:15
        env:
          POSTGRES_PASSWORD: postgres
        options: &gt;-
          --health-cmd pg_isready
```

```
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          - 5432:5432

    steps:
    - uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.11'
        cache: 'pip'

    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r python-services/api/requirements.txt pytest

    - name: Run tests
      run: pytest python-services/api/tests -v

    - name: Run linting
      run: |
        pip install flake8 black
        flake8 python-services/api --max-line-length=120
        black --check python-services/api

  build:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    permissions:
      contents: read
      packages: write

    steps:
    - uses: actions/checkout@v3

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v2

    - name: Log in to Container Registry
      uses: docker/login-action@v2
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Build and push API image
      uses: docker/build-push-action@v4
      with:
        context: ./python-services/api
        file: ./python-services/api/Dockerfile.prod
```

```
        push: true
        tags: |
          ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/api:latest
          ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/api:${{ github.sha }}
        cache-from: type=registry,ref=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/api:build
        cache-to: type=registry,ref=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/api:buildca

    - name: Build and push UI image
      uses: docker/build-push-action@v4
      with:
        context: ./web-react
        file: ./web-react/Dockerfile.prod
        push: true
        tags: |
          ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/ui:latest
          ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/ui:${{ github.sha }}

  deploy:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
    - uses: actions/checkout@v3

    - name: Configure AWS credentials
      uses: aws-actions/configure-aws-credentials@v2
      with:
        aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
        aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        aws-region: ${{ env.AWS_REGION }}

    - name: Update ECS service
      run: |
        aws ecs update-service \
          --cluster construction-ai-cluster \
          --service api \
          --force-new-deployment

    - name: Deploy to Kubernetes
      run: |
        aws eks update-kubeconfig --name construction-ai --region ${{ env.AWS_REGION }}
        kubectl set image deployment/api api=${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/ap

    - name: Notify deployment
      uses: 8398a7/action-slack@v3
      if: always()
      with:
        status: ${{ job.status }}
        text: 'Deployment ${{ job.status }}'
        webhook_url: ${{ secrets.SLACK_WEBHOOK }}
```

5 SSL/TLS & Security

SSL Setup with Let's Encrypt

File: `k8s/cert-manager.yaml`

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

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

Security Groups & Network Policies

File: `k8s/network-policy.yaml`

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: construction-ai-network-policy
  namespace: construction-ai
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: construction-ai
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          name: construction-ai
  - to:
    - podSelector: {}
    ports:
    - protocol: TCP
```

```
      port: 53
    - protocol: UDP
      port: 53
```

6 Database Backup & Recovery

PostgreSQL Backup Strategy

File: `scripts/backup-database.sh`

```
#!/bin/bash

# Configuration
DB_HOST="${RDS_ENDPOINT}"
DB_PORT="5432"
DB_NAME="construction_ai"
DB_USER="${DB_USER}"
BACKUP_DIR="/backups"
S3_BUCKET="construction-ai-backups"
TIMESTAMP=$(date +"%Y%m%d_%H%M%S")
BACKUP_FILE="$BACKUP_DIR/db_backup_$TIMESTAMP.sql.gz"

# Create backup
pg_dump -h "$DB_HOST" -p "$DB_PORT" -U "$DB_USER" -d "$DB_NAME" | gzip &gt; "$BACKUP_FILE

if [ $? -eq 0 ]; then
    echo "Database backup successful: $BACKUP_FILE"

    # Upload to S3
    aws s3 cp "$BACKUP_FILE" "s3://$S3_BUCKET/backups/"

    # Keep only last 30 days locally
    find "$BACKUP_DIR" -name "db_backup_*.sql.gz" -mtime +30 -delete

    # Send notification
    aws sns publish \
        --topic-arn "arn:aws:sns:us-east-1:ACCOUNT_ID:construction-ai-alerts" \
        --message "Database backup successful at $TIMESTAMP"
else
    echo "Database backup failed"
    aws sns publish \
        --topic-arn "arn:aws:sns:us-east-1:ACCOUNT_ID:construction-ai-alerts" \
        --message "Database backup FAILED at $TIMESTAMP"
    exit 1
fi
```

Schedule with cron:

```
# Backup daily at 2 AM UTC
0 2 * * * /usr/local/bin/backup-database.sh
```

🔄 Backup Verification

File: `scripts/verify-backup.sh`

```
#!/bin/bash

# Test restore on non-production environment
BACKUP_FILE=$1
TEST_DB="test_construction_ai"

# Create test database
createdb "$TEST_DB"

# Restore from backup
gunzip &lt; "$BACKUP_FILE" | psql "$TEST_DB"

if [ $? -eq 0 ]; then
    echo "Backup verification successful"

    # Run integrity checks
    psql "$TEST_DB" -c "SELECT COUNT(*) FROM files;"
    psql "$TEST_DB" -c "SELECT COUNT(*) FROM users;"

    # Drop test database
    dropdb "$TEST_DB"
else
    echo "Backup verification FAILED"
    exit 1
fi
```

7️⃣ Monitoring & Alerting

📊 CloudWatch Monitoring

File: `infrastructure/terraform/monitoring.tf`

```
# CloudWatch Log Group
resource "aws_cloudwatch_log_group" "api" {
  name              = "/ecs/construction-ai/api"
  retention_in_days = 30

  tags = {
    Name = "api-logs"
  }
}

# CloudWatch Alarms
resource "aws_cloudwatch_metric_alarm" "api_cpu_high" {
  alarm_name          = "construction-ai-api-cpu-high"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = 2
  metric_name         = "CPUUtilization"
```

```
  namespace           = "AWS/ECS"
  period              = 300
  statistic           = "Average"
  threshold           = 80
  alarm_description   = "Alert when API CPU is high"
  alarm_actions       = [aws_sns_topic.alerts.arn]

  dimensions = {
    ClusterName = aws_ecs_cluster.main.name
    ServiceName = aws_ecs_service.api.name
  }
}

resource "aws_cloudwatch_metric_alarm" "api_memory_high" {
  alarm_name          = "construction-ai-api-memory-high"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = 2
  metric_name         = "MemoryUtilization"
  namespace           = "AWS/ECS"
  period              = 300
  statistic           = "Average"
  threshold           = 80
  alarm_description   = "Alert when API memory is high"
  alarm_actions       = [aws_sns_topic.alerts.arn]
}

# RDS Monitoring
resource "aws_cloudwatch_metric_alarm" "rds_cpu_high" {
  alarm_name          = "construction-ai-rds-cpu-high"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = 2
  metric_name         = "CPUUtilization"
  namespace           = "AWS/RDS"
  period              = 300
  statistic           = "Average"
  threshold           = 85
  alarm_actions       = [aws_sns_topic.alerts.arn]
}

resource "aws_cloudwatch_metric_alarm" "rds_storage_low" {
  alarm_name          = "construction-ai-rds-storage-low"
  comparison_operator = "LessThanThreshold"
  evaluation_periods  = 1
  metric_name         = "FreeStorageSpace"
  namespace           = "AWS/RDS"
  period              = 300
  statistic           = "Average"
  threshold           = 10737418240  # 10 GB in bytes
  alarm_actions       = [aws_sns_topic.alerts.arn]
}

# SNS Topic for Alerts
resource "aws_sns_topic" "alerts" {
  name = "construction-ai-alerts"

  tags = {
```

```
    Name = "alerts-topic"
  }
}

resource "aws_sns_topic_subscription" "alerts_email" {
  topic_arn = aws_sns_topic.alerts.arn
  protocol  = "email"
  endpoint  = var.alert_email
}
```

8️ Load Balancing & Auto-Scaling

⚖ Application Load Balancer with Auto Scaling

File: `k8s/hpa-advanced.yaml`

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: api-hpa-advanced
  namespace: construction-ai
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: api
  minReplicas: 3
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  - type: Pods
    pods:
      metric:
        name: http_requests_per_second
      target:
        type: AverageValue
        averageValue: "1000"
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
```

```
        value: 50
        periodSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 0
      policies:
      - type: Percent
        value: 100
        periodSeconds: 15
      - type: Pods
        value: 4
        periodSeconds: 15
      selectPolicy: Max

---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: api-pdb
  namespace: construction-ai
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: api
```

9 Disaster Recovery

Failover & Recovery Procedures

File: `docs/disaster-recovery.md`

```
# Disaster Recovery Plan

## RTO &amp;amp; RPO Targets
- **RTO (Recovery Time Objective):** 1 hour
- **RPO (Recovery Point Objective):** 1 hour

## Database Failover

### Automatic Failover (Aurora)
1. Primary database becomes unavailable
2. Aurora automatically promotes read replica
3. DNS updated to point to new primary
4. Services reconnect automatically

### Manual Failover
```bash
aws rds failover-db-cluster \
  --db-cluster-identifier construction-ai-db \
  --region us-east-1
```
```

Application Recovery

Canary Deployment Rollback

```
# If new deployment has issues
kubectl set image deployment/api \
  api=ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/construction-ai/api:PREVIOUS_TAG \
  -n construction-ai

# Monitor rollback
kubectl rollout status deployment/api -n construction-ai
```

Complete Cluster Recovery

```
# Restore from backup
kubectl delete -f k8s/
terraform destroy

# Redeploy from scratch
terraform apply
kubectl apply -f k8s/

# Restore database
pg_restore -U admin -d construction_ai &lt; backup_file.sql
```

Testing Disaster Recovery

- Monthly DR drills
- Document all steps
- Measure actual RTO/RPO
- Update procedures based on learnings

```
---

## 💰 Cost Optimization

### 📉 AWS Cost Reduction Strategies

**File: `infrastructure/terraform/cost-optimization.tf`**

```hcl
# Spot Instances for non-critical workloads
resource "aws_spot_fleet_request" "batch_processing" {
  iam_fleet_role      = aws_iam_role.spot_fleet_role.arn
  replacement_strategy = "launch"
  target_capacity     = 5
  valid_from          = "2024-01-01T00:00:00Z"
  valid_until         = "2025-12-31T23:59:59Z"
```
```

```
  launch_specification {
    instance_type = "t3.large"
    image_id      = data.aws_ami.ecs.id
    spot_price    = "0.05"  # 80% cheaper than on-demand
    subnet_id     = aws_subnet.private_1.id
  }

  tags = {
    Name = "batch-processing-spot"
  }
}

# Reserved Instances for baseline capacity
resource "aws_ec2_fleet" "baseline" {
  spot_options {
    allocation_strategy = "price-capacity-optimized"
  }

  launch_template_config {
    launch_template_specification {
      version = "$Latest"
    }

    overrides {
      instance_type = "t3.medium"
      priority      = 1
    }

    overrides {
      instance_type = "t3.large"
      priority      = 2
    }
  }

  type                 = "maintain"
  target_capacity    = 10
  on_demand_target_capacity = 3  # Mix of on-demand and spot
}

# S3 Intelligent-Tiering for automatic cost optimization
resource "aws_s3_bucket_intelligent_tiering_configuration" "uploads" {
  bucket = aws_s3_bucket.uploads.id
  name   = "AutoArchive"

  tiering {
    access_tier = "ARCHIVE_ACCESS"
    days        = 90
  }

  tiering {
    access_tier = "DEEP_ARCHIVE_ACCESS"
    days        = 180
  }

  status = "Enabled"
}
```

```
# Lambda for scheduled scaling
resource "aws_lambda_function" "scale_down_offpeak" {
  filename      = "lambda_scale_down.zip"
  function_name = "construction-ai-scale-down"
  role          = aws_iam_role.lambda_role.arn
  handler       = "index.handler"
  runtime       = "python3.11"

  environment {
    variables = {
      CLUSTER_NAME = aws_ecs_cluster.main.name
      SERVICE_NAME = aws_ecs_service.api.name
    }
  }
}

# EventBridge for scheduling
resource "aws_cloudwatch_event_rule" "scale_down_schedule" {
  name                = "construction-ai-scale-down-offpeak"
  description         = "Scale down at 8 PM UTC"
  schedule_expression = "cron(0 20 * * ? *)"
}

resource "aws_cloudwatch_event_target" "scale_down_lambda" {
  rule      = aws_cloudwatch_event_rule.scale_down_schedule.name
  target_id = "ScaleDownLambda"
  arn       = aws_lambda_function.scale_down_offpeak.arn
}

# Cost monitoring dashboard
resource "aws_ce_cost_category" "construction_ai" {
  name = "construction-ai-components"

  rules {
    rule {
      dimension {
        key           = "SERVICE"
        values        = ["Amazon Elastic Container Service"]
      }
    }
    type = "REGULAR"
    value = "ECS"
  }

  rules {
    rule {
      dimension {
        key           = "SERVICE"
        values        = ["Amazon Relational Database Service"]
      }
    }
    type = "REGULAR"
    value = "RDS"
  }
```

```
  tags = {
    Project = "construction-ai"
  }
}
```

✅ Pre-Deployment Checklist

- [ ] All tests passing (unit, integration, E2E)
- [ ] Code review approved
- [ ] Security scan completed (no HIGH/CRITICAL issues)
- [ ] Database migrations tested
- [ ] Environment variables configured
- [ ] SSL certificates valid
- [ ] Backup and recovery tested
- [ ] Monitoring and alerting configured
- [ ] Team trained on deployment procedures
- [ ] Rollback plan documented
- [ ] Communication plan for maintenance window

🚀 Deployment Commands

Deploy to Kubernetes

```
# Build and push images
docker build -t api:latest python-services/api/
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-st
docker tag api:latest ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/construction-ai/api:late
docker push ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/construction-ai/api:latest

# Deploy
kubectl apply -f k8s/namespace.yaml
kubectl apply -f k8s/api-deployment.yaml
kubectl apply -f k8s/ingress.yaml

# Verify
kubectl get pods -n construction-ai
kubectl logs -f deployment/api -n construction-ai
```

Rollback

```
kubectl rollout history deployment/api -n construction-ai
kubectl rollout undo deployment/api -n construction-ai
```

Production deployment is ready! Monitor closely for the first 24 hours.