# CloudOps Demo: Multi-Tier Application on AWS EKS

## Complete DevOps Pipeline with Disaster Recovery

## Slide 1: Title Slide

**CloudOps Demo: Enterprise-Grade Multi-Tier Application** - **Subtitle**: Complete DevOps Pipeline with CI/CD, Multi-Region Deployment & Disaster Recovery - **Presenter**: [Your Name] - **Date**: [Current Date] - **Technologies**: AWS EKS, RDS, Route53, CloudWatch, Terraform, CloudFormation

## Slide 2: Project Overview

### **Objective**

Deploy a production-ready, three-tier web application with complete automation, multi-region availability, and disaster recovery capabilities.

### **Key Deliverables**

* ✅ Multi-tier application on AWS EKS
* ✅ Automated CI/CD pipeline using CodePipeline
* ✅ Multi-region deployment (Terraform + CloudFormation)
* ✅ Disaster recovery with Route53 failover
* ✅ Comprehensive monitoring with CloudWatch

## Slide 3: Architecture Overview

### **Three-Tier Architecture**

┌─────────────────┐ ┌─────────────────┐ ┌─────────────────┐  
│ Frontend │ │ Backend │ │ Database │  
│ (React) │◄──►│ (Node.js) │◄──►│ (AWS RDS) │  
│ Port: 3000 │ │ Port: 5000 │ │ Port: 5432 │  
└─────────────────┘ └─────────────────┘ └─────────────────┘

### **Technology Stack**

* **Frontend**: React 18 + Material-UI + Nginx
* **Backend**: Node.js + Express + PostgreSQL driver
* **Database**: AWS RDS PostgreSQL 15
* **Infrastructure**: Kubernetes (EKS), Docker containers

## Slide 4: Phase 1 - Multi-Tier Application Deployment

### **Task**: Deploy multi-tier Application on EKS Cluster

### **Implementation Highlights**

* **Container Strategy**: Multi-stage Docker builds for optimization
* **Orchestration**: Kubernetes manifests with proper resource management
* **Database**: AWS RDS PostgreSQL (managed service)
* **Security**: Network policies, RBAC, secrets management

### **Key Components**

* 14 Kubernetes manifests (namespace, configmaps, secrets, deployments, services)
* Horizontal Pod Autoscaler (HPA) for dynamic scaling
* Load balancer for external access
* Persistent storage for database

## Slide 5: Phase 1 - Kubernetes Architecture

### **Kubernetes Resources Deployed**

Namespace: cloudops-demo  
├── ConfigMaps: Application configuration  
├── Secrets: Database credentials  
├── Deployments:  
│ ├── Frontend (2 replicas)  
│ └── Backend (2 replicas)  
├── Services:  
│ ├── Frontend (LoadBalancer)  
│ └── Backend (ClusterIP)  
├── HPA: Auto-scaling (2-10 replicas)  
└── NetworkPolicies: Security isolation

### **Features Implemented**

* Health checks and readiness probes
* Resource limits and requests
* Rolling updates with zero downtime
* Service mesh communication

## Slide 6: Phase 2 - CI/CD Automation

### **Task**: Automate deployment using CodePipeline

### **Pipeline Components**

1. **Source**: GitHub repository integration
2. **Build**: AWS CodeBuild with custom buildspec.yml
3. **Deploy**: Automated EKS deployment with kubectl

### **buildspec.yml Highlights**

phases:  
 install: # Install kubectl, Helm, Docker  
 pre\_build: # Run tests, login to ECR  
 build: # Build Docker images, tag with commit hash  
 post\_build: # Push to ECR, deploy to EKS

### **Automation Features**

* Automated testing (frontend & backend)
* Docker image building and pushing to ECR
* Dynamic image tagging with Git commit hash
* Zero-downtime deployments with rollout status checks

## Slide 7: Phase 3 - Multi-Region Deployment

### **Task**: Expose deployment in multiple AWS Regions

### **Implementation Strategy**

* **Region 1 (us-west-2)**: Terraform Infrastructure as Code
* **Region 2 (us-east-1)**: CloudFormation templates
* **Cross-region**: Shared ECR repositories and Route53

### **Terraform Components (us-west-2)**

├── VPC with public/private subnets  
├── EKS cluster with managed node groups  
├── RDS PostgreSQL with Multi-AZ  
├── ECR repositories  
├── IAM roles and policies  
└── Load Balancer Controller

### **CloudFormation Components (us-east-1)**

* Equivalent infrastructure using CloudFormation templates
* Parameter-driven configuration
* Cross-stack references for resource sharing

## Slide 8: Phase 4 - Disaster Recovery

### **Task**: Add disaster recovery using Route53 Failover Routing

### **Failover Architecture**

Internet Users  
 │  
 ▼  
┌─────────────────┐  
│ Route53 │  
│ Health Checks │  
└─────────────────┘  
 │  
 ▼  
┌─────────────────┐ ┌─────────────────┐  
│ PRIMARY │ │ SECONDARY │  
│ us-west-2 │ │ us-east-1 │  
│ (Terraform) │ │ (CloudFormation)│  
└─────────────────┘ └─────────────────┘

### **Disaster Recovery Features**

* **Automated failover**: 30-second health check intervals
* **SNS notifications**: Real-time alerts for failures
* **Cross-region database**: RDS read replicas for data consistency
* **DNS-based routing**: Transparent user experience

## Slide 9: Phase 5 - Monitoring & Observability

### **Task**: Monitor service logs using CloudWatch Dashboard

### **Monitoring Stack**

Application Logs → CloudWatch Logs → Dashboard Widgets  
 ↓  
Health Checks → CloudWatch Alarms → SNS Notifications

### **Dashboard Components**

* **Application Logs**: Separate streams for frontend/backend
* **Performance Metrics**: CPU, memory, request latency
* **Health Checks**: Route53 endpoint monitoring
* **Custom Metrics**: Business-specific KPIs

### **Alerting Strategy**

* CloudWatch Alarms for threshold breaches
* SNS topics for email/SMS notifications
* Integration with Route53 health checks

## Slide 10: Technical Implementation Details

### **Infrastructure as Code**

* **Terraform**: 200+ lines across 8 modules
* **CloudFormation**: 300+ lines with nested stacks
* **Kubernetes**: 14 YAML manifests

### **Security Best Practices**

* Non-root containers with security contexts
* Network policies for micro-segmentation
* Secrets management with AWS Parameter Store
* IAM roles with least privilege principle

### **Performance Optimizations**

* Multi-stage Docker builds (image size reduction)
* Container resource limits and requests
* Database connection pooling
* CDN integration for static assets

## Slide 11: Key Metrics & Results

### **Deployment Metrics**

* **Build Time**: ~8 minutes (including tests)
* **Deployment Time**: ~5 minutes with zero downtime
* **Recovery Time**: <2 minutes automated failover
* **Monitoring**: 99.9% uptime SLA capability

### **Cost Optimization**

* **Spot instances**: 60% cost reduction for non-production
* **Auto-scaling**: Dynamic resource allocation
* **Reserved instances**: 40% savings for predictable workloads

### **Security Compliance**

* **Encryption**: At-rest and in-transit
* **Networking**: Private subnets with NAT gateways
* **Access control**: RBAC and security groups

## Slide 12: Challenges & Solutions

### **Challenge 1**: Container Database vs RDS

**Issue**: Initial PostgreSQL containerization not allowed **Solution**: Migrated to AWS RDS with proper security groups and networking

### **Challenge 2**: Multi-region Complexity

**Issue**: Managing consistent deployments across regions **Solution**: Parameterized templates and shared ECR repositories

### **Challenge 3**: Zero-downtime Deployments

**Issue**: Service interruption during updates **Solution**: Rolling updates with health checks and readiness probes

## Slide 13: DevOps Best Practices Implemented

### **CI/CD Excellence**

* ✅ Automated testing in pipeline
* ✅ Infrastructure as Code (GitOps)
* ✅ Container security scanning
* ✅ Rollback capabilities

### **Operational Excellence**

* ✅ Comprehensive monitoring and alerting
* ✅ Automated disaster recovery
* ✅ Documentation and runbooks
* ✅ Cost optimization strategies

### **Security & Compliance**

* ✅ Least privilege access
* ✅ Data encryption
* ✅ Network segmentation
* ✅ Audit logging

## Slide 14: Architecture Benefits

### **Scalability**

* Horizontal pod autoscaling (2-10 replicas)
* Managed node groups with automatic scaling
* Database read replicas for read-heavy workloads

### **Reliability**

* Multi-AZ database deployment
* Cross-region disaster recovery
* Health checks and automatic failover

### **Maintainability**

* Infrastructure as Code for consistency
* Automated deployments reduce human error
* Comprehensive monitoring for proactive maintenance

## Slide 15: Future Enhancements

### **Phase 6 - Advanced Features**

* **Service Mesh**: Istio for advanced traffic management
* **GitOps**: ArgoCD for declarative deployments
* **Observability**: Distributed tracing with Jaeger
* **Security**: Falco for runtime security monitoring

### **Phase 7 - Enterprise Integration**

* **Identity Management**: Integration with corporate SSO
* **Compliance**: SOC2, PCI-DSS compliance automation
* **Multi-cloud**: Deployment on Azure/GCP for vendor diversity

## Slide 16: Cost Analysis

### **Monthly Cost Breakdown (Estimated)**

* **EKS Cluster**: $72/month (control plane)
* **EC2 Instances**: $150/month (3 t3.medium nodes)
* **RDS PostgreSQL**: $45/month (db.t3.micro Multi-AZ)
* **Load Balancers**: $25/month (ALB + NLB)
* **Route53**: $5/month (health checks)
* **Total**: ~$300/month per region

### **Cost Optimization Opportunities**

* Spot instances for development environments
* Reserved instances for production (40% savings)
* Automated start/stop for non-production resources

## Slide 17: Lessons Learned

### **Technical Insights**

1. **Container Strategy**: Multi-stage builds significantly reduce image sizes
2. **Database Choice**: Managed services (RDS) provide better reliability than self-managed
3. **Monitoring**: Proactive monitoring prevents issues before they impact users

### **Process Improvements**

1. **Infrastructure as Code**: Essential for consistent, repeatable deployments
2. **Automated Testing**: Catches issues early in the development cycle
3. **Documentation**: Critical for team knowledge sharing and onboarding

## Slide 18: Demo Walkthrough

### **Live Demonstration**

1. **Application Access**: Show working three-tier application
2. **CI/CD Pipeline**: Trigger deployment via Git commit
3. **Failover Test**: Simulate primary region failure
4. **Monitoring**: Real-time CloudWatch dashboard
5. **Recovery**: Automatic failback to primary region

### **Key URLs**

* Application: https://cloudops-demo.yourdomain.com
* Pipeline: AWS CodePipeline console
* Monitoring: CloudWatch dashboard

## Slide 19: Knowledge Transfer

### **Documentation Delivered**

* ✅ Complete README with setup instructions
* ✅ Architecture diagrams and decision records
* ✅ Deployment runbooks and troubleshooting guides
* ✅ Cost optimization recommendations

### **Skills Developed**

* Container orchestration with Kubernetes
* Infrastructure automation with Terraform/CloudFormation
* CI/CD pipeline design and implementation
* Disaster recovery planning and testing
* Cloud-native monitoring and observability

## Slide 20: Q&A and Next Steps

### **Questions & Discussion**

* Technical implementation details
* Scaling considerations
* Security best practices
* Cost optimization strategies

### **Next Steps**

1. **Production Readiness**: Security audit and performance testing
2. **Team Training**: Knowledge transfer sessions
3. **Expansion**: Additional regions and services
4. **Optimization**: Continuous improvement based on metrics

### **Contact Information**

* **Project Repository**: [GitHub URL]
* **Documentation**: [Wiki/Confluence URL]
* **Support**: [Team contact information]

## Additional Slides: Technical Deep Dives

### Slide 21: Kubernetes Manifest Examples

# Frontend Deployment snippet  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
 name: frontend-deployment  
 namespace: cloudops-demo  
spec:  
 replicas: 2  
 selector:  
 matchLabels:  
 app: frontend  
 template:  
 spec:  
 containers:  
 - name: frontend  
 image: your-registry/cloudops-frontend:latest  
 ports:  
 - containerPort: 3000  
 resources:  
 requests:  
 memory: "128Mi"  
 cpu: "100m"  
 limits:  
 memory: "256Mi"  
 cpu: "200m"

### Slide 22: Terraform Module Structure

terraform/  
├── main.tf # Main configuration  
├── variables.tf # Input variables  
├── outputs.tf # Output values  
└── modules/  
 ├── rds/ # RDS PostgreSQL module  
 ├── ecr/ # Container registry module  
 ├── iam/ # IAM roles and policies  
 ├── load-balancer-controller/  
 └── route53-failover/ # DNS failover module

### Slide 23: CloudFormation Template Structure

# CloudFormation nested stack example  
AWSTemplateFormatVersion: '2010-09-09'  
Description: 'CloudOps Demo - Complete Infrastructure'  
  
Parameters:  
 EnvironmentName:  
 Type: String  
 Default: cloudops-demo  
  
Resources:  
 VPCStack:  
 Type: AWS::CloudFormation::Stack  
 Properties:  
 TemplateURL: !Sub 'https://s3.amazonaws.com/templates/vpc.yaml'  
 Parameters:  
 EnvironmentName: !Ref EnvironmentName  
  
 EKSStack:  
 Type: AWS::CloudFormation::Stack  
 Properties:  
 TemplateURL: !Sub 'https://s3.amazonaws.com/templates/eks.yaml'  
 Parameters:  
 VPCId: !GetAtt VPCStack.Outputs.VPC