Multi-Cloud Kubernetes Platform

A comprehensive web-based platform for provisioning and managing Kubernetes clusters across multiple cloud providers through a unified interface. Built with modern web technologies and infrastructure-as-code principles, this platform simplifies multi-cloud Kubernetes deployment while maintaining enterprise-grade security and reliability.

Platform Screenshot

🌟 Features

Multi-Cloud Support

- **AWS Integration**: Complete EKS cluster provisioning with VPC, security groups, and IAM roles
- **Azure Integration**: Full AKS cluster creation with virtual networks and resource groups
- **Unified Interface**: Single UI for managing clusters across both cloud providers

User Experience

- **Intuitive Web Interface**: Modern React-based UI with responsive design
- **Real-time Status Updates**: Live cluster provisioning progress tracking
- **Role-based Access Control**: Admin and user roles with appropriate permissions
- **Professional Dashboard**: Clean, modern interface with comprehensive cluster overview

Security & Reliability

- **Authentication System**: Secure session-based authentication with role management
- **Input Validation**: Comprehensive validation and error handling
- **Secure Communication**: HTTPS enforcement and CORS configuration

- **Audit Logging**: Complete activity tracking and monitoring

Infrastructure as Code

- **Terraform Modules**: Reusable, maintainable infrastructure definitions
- **State Management**: Remote state storage configuration
- **Parameterized Deployments**: Configurable cluster specifications
- **Version Control**: Infrastructure changes tracked in Git

🏗️ Architecture

The platform follows a modern three-tier architecture:

Plain Text

```
┌─────────────────┐     ┌─────────────────┐     ┌─────────────────┐
│    Frontend     │     │     Backend     │     │ Infrastructure  │
│    (React)      │◄───►│    (Flask)      │◄───►│  (Terraform)    │
│                 │     │                 │     │                 │
│ • User Interface│     │ • REST API      │     │ • AWS Modules   │
│ • Authentication│     │ • Authentication│     │ • Azure Modules │
│ • State Mgmt    │     │ • Cluster Mgmt  │     │ • State Backend │
└─────────────────┘     └─────────────────┘     └─────────────────┘
```

Component Overview

- **Frontend**: React application with TypeScript support, responsive design, and modern UI components
- **Backend**: Flask REST API with SQLAlchemy ORM, session authentication, and asynchronous task processing
- **Infrastructure**: Terraform modules for AWS EKS and Azure AKS with remote state management
- **Database**: SQLite for development, PostgreSQL support for production

🚀 Quick Start

Prerequisites

- **Python 3.8+** for backend development
- **Node.js 16+** for frontend development
- **Terraform 1.0+** for infrastructure provisioning
- **Git** for version control
- **AWS Account** with appropriate IAM permissions
- **Azure Account** with subscription and service principal

Installation

1. **Clone the repository**
2. **Set up the backend**
3. **Set up the frontend**
4. **Configure environment variables**
5. **Initialize the database**

Running the Application

1. **Start the backend**
2. **Build and serve the frontend**
3. **Access the application**
   Open your browser and navigate to `http://localhost:5000`

Default Credentials

- **Username**: admin

- **Password**: admin123

> ⚠️ **Security Note**: Change the default credentials immediately after first login in production environments.

📖 Usage Guide

Creating Your First Cluster

1. **Login** to the platform using the provided credentials
2. **Navigate** to the dashboard and click "Create Cluster"
3. **Configure** your cluster:
   - **Cluster Name**: Choose a descriptive name
   - **Cloud Provider**: Select AWS or Azure
   - **Region**: Choose your preferred region
   - **Kubernetes Version**: Select from available versions
   - **Node Count**: Specify the number of worker nodes
   - **Instance Type**: Choose appropriate compute resources
4. **Submit** the configuration to begin provisioning
5. **Monitor** the real-time progress on the dashboard

Managing Clusters

- **View Details**: Click on any cluster to see comprehensive information
- **Monitor Status**: Real-time status updates during provisioning
- **Download Config**: Retrieve kubeconfig files for kubectl access
- **Delete Clusters**: Admin users can remove clusters with confirmation

User Management

- **Profile Settings**: Update passwords and contact information
- **Role Assignment**: Administrators can manage user roles
- **Audit Logs**: Review user activity and access patterns

🔧 Configuration

Environment Variables

The platform uses environment variables for configuration management:

Bash

```bash
# Flask Configuration
FLASK_ENV=production
SECRET_KEY=your-super-secret-key-here
DATABASE_URL=sqlite:///app.db

# AWS Configuration
AWS_ACCESS_KEY_ID=your-aws-access-key
AWS_SECRET_ACCESS_KEY=your-aws-secret-key
AWS_DEFAULT_REGION=us-east-1

# Azure Configuration
AZURE_CLIENT_ID=your-azure-client-id
AZURE_CLIENT_SECRET=your-azure-client-secret
AZURE_TENANT_ID=your-azure-tenant-id
AZURE_SUBSCRIPTION_ID=your-azure-subscription-id

# Application Settings
ALLOWED_HOSTS=localhost,your-domain.com
CORS_ORIGINS=https://your-domain.com
```

Cloud Provider Setup

AWS Configuration

1. **Create IAM User** with programmatic access

2. **Attach Policy** with EKS, EC2, and IAM permissions
3. **Generate Access Keys** and configure in environment variables
4. **Set up S3 Backend** for Terraform state (recommended for production)

Azure Configuration

1. **Create Service Principal** using Azure CLI
2. **Assign Contributor Role** to the service principal
3. **Configure Environment Variables** with client credentials
4. **Set up Storage Account** for Terraform state (recommended for production)

Terraform State Management

For production deployments, configure remote state storage:

Plain Text

```
terraform {
  backend "s3" {
    bucket         = "your-terraform-state-bucket"
    key            = "multicloud-k8s/terraform.tfstate"
    region         = "us-east-1"
    dynamodb_table = "terraform-locks"
    encrypt        = true
  }
}
```

🔒 Security

Authentication & Authorization

- **Session-based Authentication**: Secure session management with appropriate timeouts
- **Role-based Access Control**: Granular permissions for different user types

- **Password Security**: Bcrypt hashing with salt rounds
- **CSRF Protection**: Cross-site request forgery prevention

Data Protection

- **Encryption at Rest**: Sensitive database fields are encrypted
- **Encryption in Transit**: All communications use HTTPS/TLS
- **Credential Management**: Cloud provider credentials stored securely
- **Audit Logging**: Comprehensive activity tracking

Network Security

- **Firewall Configuration**: Minimal port exposure with proper rules
- **CORS Configuration**: Controlled cross-origin resource sharing
- **Rate Limiting**: Protection against abuse and DoS attacks
- **SSL/TLS**: Strong encryption for all web traffic

🚀 Deployment

Production Deployment

1. **Server Preparation**
2. **Application Setup**
3. **Service Configuration**
4. **Nginx Configuration**
5. **SSL Configuration**

Docker Deployment

For containerized deployment, use the provided Docker configurations:

Plain Text

```
# Backend Dockerfile
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python", "src/main.py"]
```

YAML

```yaml
# docker-compose.yml
version: '3.8'
services:
  backend:
    build: ./multicloud_k8s_api
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=production
    volumes:
      - ./data:/app/data

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
    depends_on:
      - backend
```

📊 Monitoring & Maintenance

Health Checks

The platform includes built-in health check endpoints:

Bash

```bash
# Application health
curl -X GET http://localhost:5000/health

# API status
curl -X GET http://localhost:5000/api/auth/status
```

Logging

Comprehensive logging is configured for monitoring and troubleshooting:

Python

```python
# Application logs
tail -f /opt/multicloud-k8s/logs/app.log

# System logs
sudo journalctl -u multicloud-k8s -f

# Nginx logs
sudo tail -f /var/log/nginx/access.log
```

Backup Strategy

Regular backups should include:

- **Database**: SQLite database file or PostgreSQL dump
- **Configuration**: Environment variables and application settings
- **Terraform State**: Remote state files and lock tables
- **SSL Certificates**: Let's Encrypt certificates and keys

Performance Monitoring

Monitor key metrics for optimal performance:

- **Response Times**: API endpoint performance
- **Resource Usage**: CPU, memory, and disk utilization

- **Database Performance**: Query execution times
- **Error Rates**: Application and infrastructure errors

🔧 Development

Development Environment

1. **Backend Development**
2. **Frontend Development**
3. **Testing**

Code Structure

Plain Text

```
multicloud-k8s-platform/
├── multicloud_k8s_api/          # Backend Flask application
│   ├── src/
│   │   ├── models/               # Database models
│   │   ├── routes/               # API endpoints
│   │   ├── static/               # Static files (built frontend)
│   │   └── main.py               # Application entry point
│   ├── requirements.txt          # Python dependencies
│   └── .env.example             # Environment configuration template
├── multicloud-k8s-ui/           # Frontend React application
│   ├── src/
│   │   ├── components/           # React components
│   │   ├── hooks/                # Custom React hooks
│   │   ├── lib/                  # Utility functions
│   │   └── main.jsx              # Application entry point
│   ├── package.json              # Node.js dependencies
│   └── vite.config.js            # Build configuration
├── terraform/                    # Infrastructure as Code
│   ├── modules/
│   │   ├── aws_kubernetes/       # AWS EKS module
│   │   └── azure_kubernetes/     # Azure AKS module
│   └── backend.tf                # State backend configuration
└── docs/                         # Documentation
    ├── architecture_design.md
    ├── deployment_guide.md
    └── platform_documentation.md
```

Contributing

1. **Fork** the repository
2. **Create** a feature branch (`git checkout -b feature/amazing-feature`)
3. **Commit** your changes (`git commit -m 'Add amazing feature'`)
4. **Push** to the branch (`git push origin feature/amazing-feature`)
5. **Open** a Pull Request

Coding Standards

- **Python**: Follow PEP 8 style guidelines
- **JavaScript/React**: Use ESLint and Prettier for consistent formatting
- **Documentation**: Write clear, comprehensive documentation for new features
- **Testing**: Include unit tests for new functionality

🐛 Troubleshooting

Common Issues

Backend Service Issues

Problem: Backend service fails to start
Solution: Check logs and verify dependencies

Bash

```bash
# Check service status
sudo systemctl status multicloud-k8s

# View logs
```

```
sudo journalctl -u multicloud-k8s -n 50

# Verify Python dependencies
cd /opt/multicloud-k8s/backend
source venv/bin/activate
pip check
```

Database Connection Issues

Problem: Database connection errors

Solution: Verify database file permissions and path

Bash

```
# Check database file
ls -la /opt/multicloud-k8s/backend/src/database/

# Test database connection
cd /opt/multicloud-k8s/backend
source venv/bin/activate
python -c "from src.models.cluster import db; print('Database connection successful')"
```

Terraform Execution Issues

Problem: Terraform commands fail

Solution: Verify cloud provider credentials and permissions

Bash

```
# Test AWS credentials
aws sts get-caller-identity

# Test Azure credentials
az account show

# Validate Terraform configuration
cd terraform
terraform validate
```

Getting Help

- **Documentation**: Comprehensive guides in the `docs/` directory
- **Issues**: Report bugs and request features on GitHub
- **Community**: Join our community discussions
- **Support**: Contact the development team for enterprise support

📄 License

This project is licensed under the MIT License - see the LICENSE file for details.

🙏 Acknowledgments

- **HashiCorp Terraform** for infrastructure as code capabilities
- **AWS** and **Microsoft Azure** for cloud platform support
- **React** and **Flask** communities for excellent frameworks
- **Open Source Community** for inspiration and best practices

📞 Support

For support and questions:

- **Documentation**: Check the comprehensive documentation in the `docs/` directory
- **Issues**: Create an issue on GitHub for bugs and feature requests
- **Email**: Contact the development team at support@example.com
- **Community**: Join our community discussions and forums

Built with ❤️ by the Multi-Cloud Kubernetes Platform Team

Simplifying multi-cloud Kubernetes deployment, one cluster at a time.