

Project Overview

Our Cloud Cost Optimizer Dashboard is a web based application designed to monitor performance and costs of cloud instances. Our product aims to provide a centralized platform to help users make better decisions regarding their cloud infrastructure across different providers. Users can track resource usage, monitor system health, identify cost anomalies, and receive recommendations to reduce unnecessary spending. With configurable alerts, automated reports, and intelligent optimization suggestions

TEAM

- Created by Team Cloud for CIS4951 - Capstone II course at Florida International University.
- Contributors
 - Daniel Gonzalez - Project Leader / Full Stack Developer
 - Sandy Aguilar - Full Stack Developer
 - Isabel Ruiz - Full Stack Developer
 - Favio Valdes - Full Stack Developer
 - Yasser Blanco Montequin - Full Stack Developer

FEATURES

Authentication & Security

- JWT-based authentication with secure token storage with 8-hour expiry
- Role-based access control (Admin/Viewer roles)
- Password reset functionality with token-based recovery
- Auto-logout on 401 responses for expired sessions
- Protected routes that require authentication
- Modular auth components for better maintainability
- Password hashing with salt
- Environment variables for sensitive data
- Mock data mode for safe demos

Dashboard Capabilities

- Multi-cloud support for AWS, GCP, and Azure
- Real-time metrics monitoring with auto-refresh (30-second intervals)
- Month-to-date cost tracking by service and project
- Interactive spending visualizations using Recharts
- Provider-specific dashboards with detailed breakdowns
- Consistent provider ordering across all dashboard sections (AWS → AZURE → GCP)
- Color-coded pie chart matching provider icons

Cost Intelligence Features

- Anomaly Detection - Identifies unusual spending patterns with severity levels and recommendations
- Cost Forecasting - 7-day cost predictions with confidence intervals and trend analysis
- Optimization Recommendations - Actionable suggestions with potential savings estimates
- Cost Alerts - Real-time threshold notifications with severity-based filtering
- Budget Tracking - Visual budget vs actual spending with projections and at-risk indicators
- Services Breakdown - Detailed cost analysis by category with expandable service lists

USAGE

1. User Registration

- a. Navigate to login page
- b. Click "Create account"
- c. Enter username and password
- d. Optionally add email for password recovery
- e. Select role (Viewer or Admin)
- f. Account is created instantly

2. Login Process

- a. Enter username and password
- b. Click "Sign In"
- c. JWT token is stored in browser
- d. Redirected to dashboard

3. Password Reset

- a. Click "Forgot password?" on login
- b. Enter username or email
- c. Receive reset token (displayed in dev mode)
- d. Click "I already have a token"
- e. Enter token and new password
- f. Password is updated

4. Dashboard Features

- a. Overview Tab
 - i. Total costs across all providers
 - ii. Provider breakdown cards (AWS → Azure → GCP)
 - iii. Services Breakdown with expandable categories
 - iv. Interactive pie chart for spending distribution
- b. Provider Tabs (AWS, Azure, GCP)
 - i. CPU Usage and Instances Monitored metrics

- ii. MTD Total Cost and Active Services count
 - iii. Scrollable Month-to-Date Costs table by service
- c. Insights & Alerts Tab
 - i. Alerts Panel: Real-time cost alerts with severity filtering (Critical, High, Warning, Low)
 - ii. Recommendations Panel: Cost optimization suggestions with potential savings, effort level, and impact
 - iii. Anomaly Detection Panel: Unusual spending patterns with deviation percentages and recommendations
- d. Budgets & Forecast Tab
 - i. Cost Forecast Chart: 7-day cost predictions with confidence intervals, trend indicators (Actual vs Forecast)
 - ii. Budget Tracker: Budget vs actual spending with By Provider / By Category toggle, utilization percentages, and at-risk indicators

System Requirements

- Python 3.8 or higher
- Node.js 16.x or higher
- npm 8.x or higher
- Git for version control

Start Application

1. Clone the repository

```
git clone <repository-url>  
cd Cloud-Cost-Optimizer-Dashboard
```

2. Start the backend. Backend will automatically:
 - a. Create the SQLite database on first run
 - b. Initialize required tables.
 - c. Serve mock data or real live data based on configured .env file
 - d. Run on <http://localhost:5000>

```
# Navigate to backend directory
cd backend

# Install Python dependencies
pip install -r requirements.txt

# Start the Flask server
python app.py
```

3. Start the Frontend

- a. Frontend will run on <http://localhost:5173>

```
# Open new terminal and navigate to frontend
cd frontend

# Install Node dependencies
npm install

# Start development server
npm run dev
```

4. Create an account

- a. Open browser to <http://localhost:5173> (frontend)
- b. Click create account on login page
- c. Fill in:
 - i. Username*
 - ii. Password*
 - iii. Email (optional for password recovery)
 - iv. Role (select Admin for full access)
 - v. Note: Only one user can be admin, once an user is created with admin role no more will be permitted

5. Configuration: Edit the .env file in the backend directory with the following option

- a. Mock Data Mode: By default the .env file includes `USE MOCK_DATA=true`
- b. Real Data Mode: Set the mock data variable to false and add cloud provider credential using the following format:

```
# Edit backend/.env file
USE MOCK_DATA=false

# AWS Credentials
AWS_ACCESS_KEY_ID=your-access-key
AWS_SECRET_ACCESS_KEY=your-secret-key
AWS_REGION=us-east-1

# GCP Credentials
GCP_PROJECT_ID=your-project-id
GCP_CREDENTIALS_PATH=/path/to/service-account.json

# Azure Credentials
AZURE_SUBSCRIPTION_ID=your-subscription-id
AZURE_TENANT_ID=your-tenant-id
AZURE_CLIENT_ID=your-client-id
AZURE_CLIENT_SECRET=your-client-secret
```

Common Troubleshooting

1. Backend Issues

a. Port 5000 already in use:

```
# Find process using port 5000
lsof -i :5000 # Mac/Linux
netstat -ano | findstr :5000 # Windows

# Or use different port
python app.py --port 5001
```

b. Database errors:

```
# Delete database and let it recreate
rm instance/app.db # Mac/Linux
del instance\app.db # Windows
python app.py
```

c. No cloud data showing:

i. Check if USE MOCK_DATA=true in .env

- ii. If set to false verify that cloud credentials are correct and use the specified format

2. Frontend Issues

- a. Cannot connect to backend:
 - i. Verify backend is running on port 5000
 - ii. Check browser console for errors
 - iii. Ensure both servers are running
- b. Login not working
 - i. Clear browser localStorage
 - ii. Check network tab for API errors
 - iii. Verify backend is responding

3. Quick Reset:

```
# Backend
cd backend
rm -rf instance/ # Remove database
python app.py

# Frontend (new terminal)
cd frontend
npm install
npm run dev
```

Mock Data Details

- 3 cloud providers (AWS, Azure, GCP)
- 140+ cloud services across 8 categories:
 - AWS: 50+ services (EC2, Lambda, S3, RDS, etc.)
 - Azure: 56+ services (VMs, Functions, Blob Storage, etc.)
 - GCP: 53+ services (Compute Engine, Cloud Functions, etc.)
- Categories: Compute, Storage, Database, Networking, Analytics, Machine Learning, Security, Management
- Simulated anomalies with severity levels
- 7-day cost forecasts with confidence intervals
- Budget tracking with utilization metrics
- Cost optimization recommendations with savings estimates
- Real-time alerts with severity filtering
- Perfect for demos and development

UI/UX DESIGN

- Provider Color Scheme
 - AWS: Orange (#f59e0b) with  icon
 - Azure: Blue (#3b82f6) with  icon
 - GCP: Gray (#94a3b8) with  icon
- Provider Ordering
 - Navigation tabs: AWS → AZURE → GCP
 - Pie chart legend: AWS → AZURE → GCP
 - Provider cards: AWS → AZURE → GCP
- Category Color Scheme
 - Compute: Red (#ef4444)
 - Storage: Orange (#f97316)
 - Database: Yellow (#eab308)
 - Networking: Green (#22c55e)
 - Analytics: Teal (#14b8a6)
 - Machine Learning: Blue (#3b82f6)
 - Security: Purple (#8b5cf6)
 - Management: Pink (#ec4899)

CONTRIBUTIONS

1. Fork the repository
2. Create feature branch (git checkout -b feature/NewFeature)
3. Commit changes (git commit -m 'Add NewFeature')
4. Push to branch (git push origin feature/NewFeature)
5. Open Pull Request
6. Development Best Practices:
 - a. Keep mock data mode enabled for development
 - b. Test with real credentials only in secure environment
 - c. Follow existing code structure
 - d. Update documentation for new features
 - e. Maintain consistent provider ordering (AWS → AZURE → GCP)
 - f. Use the established color scheme for providers

LICENSE

This project is licensed under the MIT License