Continuous Integration/Continuous Delivery (CI/CD)

PROG 8860

MindfulMinutes - CI/CD Pipeline Implementation

Group Members:

- Cibi Sharan Cholarani (9015927)
- Nikhil Shankar Chirakkal Sivasankaran (9026254)

Application Overview

MindfulMinutes is a wellness application built using the MEAN stack (MongoDB, Express.js, Angular, Node.js).

Core Features:

- Daily motivational quotes
- Guided breathing exercises
- Personal journaling
- Activity tracking
- JWT-based authentication

Tech Stack:

- Frontend: Angular
- Backend: Node.js + Express
- Database: MongoDB Atlas
- Authentication: JWT

Infrastructure Architecture

AWS Services:

- **Frontend:** S3 + CloudFront
- **Backend:** EC2 with Docker + API Gateway
- **Container Registry:** ECR
- **Monitoring:** Prometheus + Grafana
- **Infrastructure as Code:** Terraform

Environments:

- Dev (dev branch)
- Production (main branch)

Pipeline Workflow Architecture

Infrastructure Pipeline

```
Trigger (terraform/** changes)
    ↓
Terraform Init
    ↓
Workspace Selection (dev/prod)
    ↓
Terraform Plan
    ↓
Terraform Apply
    ↓
Deploy: EC2, S3, CloudFront, Security Groups, API Gateway
```

Backend Pipeline

```
Trigger (backend/** changes)
    ↓
Test Stage
    ├── npm ci
    ├── Run tests
    └── Upload coverage
    ↓
Build & Deploy Stage
    ├── Build Docker image
    ├── Push to ECR
    ├── Deploy to EC2 via SSM
    └── Start monitoring stack
```

Frontend Pipeline

```
Trigger (frontend/** changes)
    ↓
Test Stage
    ├── npm ci
    └── Run tests
    ↓
Build & Deploy Stage
    ├── Get API Gateway URL
    ├── Inject backend URL
    ├── Build Angular (prod/dev config)
    ├── Sync to S3
    └── Invalidate CloudFront cache
```

Pipeline Design

Stages & Tools

1. Infrastructure Deployment

- **Tool:** Terraform, GitHub Actions
- **Stages:** Init → Plan → Apply
- **Resources:** EC2, S3, CloudFront, API Gateway, VPC, Security Groups

2. Backend CI/CD

- **CI Stage:**
  - Test execution with coverage
  - Dependency installation
- **CD Stage:**
  - Docker containerization
  - ECR push
  - EC2 deployment via AWS SSM
  - Monitoring setup (Prometheus, Grafana, cAdvisor)

3. Frontend CI/CD

- **CI Stage:**
  - Angular unit tests
  - Dependency validation
- **CD Stage:**
  - Environment-specific builds
  - Dynamic API URL injection
  - S3 deployment
  - CloudFront invalidation

4. Monitoring & Observability

- **Prometheus:** Metrics collection
- **Grafana:** Dashboards (port 3003)
- **cAdvisor:** Container metrics
- **Node Exporter:** System metrics
- **Coverage Reports:** Hosted on port 8080

Deployment Flow

Frontend:

1. Build Angular app with environment configuration
2. Deploy static assets to S3
3. Distribute via CloudFront CDN
4. Invalidate cache for instant updates

Backend:

1. Run automated tests
2. Build and tag Docker image
3. Push to Amazon ECR
4. Deploy to EC2 using Docker Compose
5. Start monitoring stack with backend

Challenges & Solutions

1. GitHub Repository Secret Access

Challenge: Organization collaborators couldn't access repository secrets needed for CI/CD workflows.

Solution: Created a GitHub organization, migrated the repository, and added team members as repository admins with full secret access.

---

2. Hardcoded Backend URLs

Challenge: Frontend code contained hardcoded backend API URLs, preventing environment-specific deployments.

Solution: Created three environment configuration files (`environment.ts`, `environment.development.ts`, `environment.production.ts`) with placeholder values (`API_URL_PLACEHOLDER`). The deployment pipeline dynamically injects the correct API Gateway URL based on the target environment.

---

3. CloudFront HTTPS to EC2 HTTP Incompatibility

Challenge: CloudFront serves content over HTTPS, but direct EC2 backend communication used HTTP, causing mixed content security errors.

Solution: Implemented AWS API Gateway as an intermediary layer, which provides native HTTPS endpoints and forwards requests to the backend EC2 instance.

---

4. MongoDB Atlas Connection Failure

Challenge: Backend on EC2 couldn't connect to MongoDB Atlas, while local development worked fine.

Solution: Added the EC2 instance's public IP address to MongoDB Atlas IP whitelist to allow network access.

---

5. Connection Loss on Redeployment

Challenge: MongoDB connection failed after EC2 redeployments because the public IP changed dynamically.

Solution: Allocated an Elastic IP to the EC2 instance to ensure a static IP address, then whitelisted this permanent IP in MongoDB Atlas.

Key Achievements

- Automated multi-environment deployment (dev/prod)
- Path-based workflow triggers for efficient builds
- Containerized backend with monitoring stack
- Infrastructure as Code with Terraform
- Full test coverage reporting
- Zero-downtime deployments via CloudFront