Simple GitHub Actions Deployment Setup

This document explains how to set up the simplified GitHub Actions workflow for automated deployment of the RBarros application.

🚀 Overview

The GitHub Actions workflow uses a **simple deployment strategy** that:

- ☑ Connects to your server via SSH
- ☑ Pulls the latest code from your repository
- ☑ Passes GitHub secrets as environment variables
- ☑ Rebuilds and restarts Docker containers
- ☑ Uses basic username/password authentication

Simple and straightforward - no .env files needed, secrets managed in GitHub!

📋 Prerequisites

1. **Production Server**: A Linux server with Docker and Docker Compose installed
2. **SSH Access**: Username/password access to your production server
3. **GitHub Repository**: Admin access to set up secrets

🔐 Required GitHub Secrets

Go to your GitHub repository → Settings → Secrets and variables → Actions → New repository secret

Server Connection Secrets

- `SERVER_HOST`: Your production server IP address or domain
- `SERVER_USERNAME`: SSH username (e.g., `ubuntu`, `root`, or your user)
- `SERVER_PASSWORD`: SSH password
- `SERVER_PORT`: SSH port (optional, defaults to 22)

Database Secrets

- `DB_HOST`: Database host (`database` for Docker Compose setup)
- `DB_USER`: Database username
- `DB_PASSWORD`: Database password
- `DB_NAME`: Database name (e.g., `rbarrosassurance`)
- `MYSQL_ROOT_PASSWORD`: MySQL root password

Application Secrets

- `SECRET_KEY`: JWT secret key for authentication
- `SECRET_KEY_REFRESH_TOKEN`: JWT refresh token secret
- `SENDGRID_API_KEY`: SendGrid API key for email services

- `WEBHOOK_SECRET`: Webhook secret for secure communications

Frontend Configuration

- `VUE_APP_API_URL`: Frontend API URL (e.g., `https://yourdomain.com/api`)

🏗 Production Server Setup

1. Install Docker & Docker Compose

```
# Ubuntu/Debian
sudo apt update
sudo apt install docker.io docker-compose-plugin

# Start Docker service
sudo systemctl enable docker
sudo systemctl start docker

# Add user to docker group (optional)
sudo usermod -aG docker $USER
newgrp docker
```

2. Clone Your Repository (One Time Only)

```
# Clone your repository to the server
cd ~
git clone --recurse-submodules https://github.com/yourusername/rbarros-
deployment.git
cd rbarros-deployment
```

3. Configure Firewall (Optional)

```
# Allow SSH
sudo ufw allow 22

# Allow HTTP/HTTPS (if using nginx)
sudo ufw allow 80
sudo ufw allow 443

# Enable firewall
sudo ufw enable
```

🚀 How It Works

When you push code to the `main` branch, GitHub Actions will:

1. **Connect to Server**: SSH into your production server
2. **Pass Secrets**: All GitHub secrets are passed as environment variables
3. **Update Code**: Run `git pull --recurse-submodules origin main` to get latest changes
4. **Restart Services**: Stop and rebuild Docker containers with new code and secrets
5. **Confirm Success**: Show deployment complete message

No .env file needed - all secrets are passed directly from GitHub!

🛠️ Manual Operations

Check Status

```
cd ~/rbarros-deployment
docker-compose -f docker-compose.prod.yml ps
```

View Logs

```
docker-compose -f docker-compose.prod.yml logs -f
```

Manual Deploy (with secrets)

```
cd ~/rbarros-deployment
git pull --recurse-submodules origin main

# You'll need to set environment variables manually for manual deploy
export NODE_ENV=production
export DB_HOST=your_db_host
export DB_USER=your_db_user
# ... set all other variables ...

docker-compose -f docker-compose.prod.yml down
docker-compose -f docker-compose.prod.yml up -d --build
```

🐛 Troubleshooting

SSH Connection Issues

- Verify server IP, username, and password in GitHub secrets
- Check if SSH service is running: `sudo systemctl status ssh`
- Test connection manually: `ssh username@server-ip`

Git Pull Issues

- Make sure repository is accessible from server

- Check if you need to set up SSH keys for Git access
- Verify the repository URL is correct
- Ensure submodules are properly initialized

Docker Issues

- Check if Docker is running: `sudo systemctl status docker`
- Verify docker-compose.prod.yml file exists and uses environment variables
- Check container logs for errors: `docker-compose -f docker-compose.prod.yml logs`

Environment Variable Issues

- Ensure all required GitHub secrets are set
- Check that docker-compose.prod.yml references environment variables correctly
- Verify secrets are being passed with the `envs:` parameter in workflow

📊 What Gets Deployed

The workflow assumes you have:

- A `docker-compose.prod.yml` file that uses environment variables (not .env file)
- All necessary secrets configured in GitHub repository settings
- Docker containers configured to run your application with passed environment variables

That's it! Simple, straightforward deployment with secrets managed securely in GitHub!