

Evolution Go API

Evolution Go API is a high-performance, lightweight WhatsApp API implementation written in Go. Built with Go's standard library and modern best practices, it provides a robust and efficient solution for WhatsApp integration using the [whatsmeow library](#) - a Go implementation of the WhatsApp web multidevice API.

This project is part of the Evolution ecosystem, offering a Go-based alternative that focuses on performance, simplicity, and reliability. Evolution Go API is designed for developers who need a fast, scalable WhatsApp API solution with minimal dependencies.

Built with whatsmeow

This project uses [whatsmeow](#), a Go library for the WhatsApp web multidevice API created by [tulir](#). whatsmeow provides the core WhatsApp protocol implementation that powers this API.

💡 Features

- **High Performance:** Built with Go for maximum performance and minimal resource usage
- **Standard Library:** Uses Go's standard library `net/http` with the new ServeMux (Go 1.22+)
- **WhatsApp Integration:** Full WhatsApp Web API support through whatsmeow
- **RESTful API:** Clean, well-documented REST endpoints
- **Real-time Events:** WebSocket support for real-time message handling
- **Message Storage:** Optional PostgreSQL integration for message persistence
- **Media Support:** Complete media handling (images, videos, audio, documents)
- **QR Code Generation:** Built-in QR code generation for device pairing
- **Docker Support:** Ready-to-use Docker configuration
- **Swagger Documentation:** Auto-generated API documentation
- **Logging:** Comprehensive logging with configurable levels
- **Middleware Support:** Authentication, validation, and custom middleware
- **Event System:** Support for webhooks, AMQP, and WebSocket events

🛠️ Technology Stack

- **Language:** Go 1.24+
- **HTTP Framework:** Standard library `net/http` with ServeMux
- **WhatsApp Library:** [whatsmeow](#) - Go library for WhatsApp web multidevice API
- **Database:** PostgreSQL (optional)
- **Documentation:** Swagger/OpenAPI
- **Containerization:** Docker
- **Message Queue:** AMQP/RabbitMQ support
- **Storage:** MinIO/S3 compatible storage

Prerequisites

- Go 1.24 or higher
- PostgreSQL (optional, for message storage)
- Docker (optional, for containerized deployment)

Quick Start

Using Docker (Recommended)

```
# Clone the repository
git clone https://git.evoai.app/Evolution/evolution-go.git
cd evolution-go

# Build and run with Docker
make docker-build
make docker-run
```

Local Development

```
# Clone the repository
git clone https://git.evoai.app/Evolution/evolution-go.git
cd evolution-go

# Setup complete environment (installs deps + generates swagger)
make setup

# Copy environment file and configure
cp .env.example .env
# Edit .env with your configuration

# Run in development mode
make dev

# Or run with hot reload (requires air)
make watch
```

Tips:

- Run `make help` to see all available commands
- See [COMMANDS.md](#) for detailed command guide and workflows

Using Make Commands

The project includes a comprehensive Makefile with useful commands:

```
# Show all available commands
make help

# Development
make dev          # Run in development mode
make run          # Run in production mode
make watch        # Run with hot reload (requires air)
```

```

# Build
make build           # Build for current platform
make build-all       # Build for all platforms
make install         # Build and install to GOPATH

# Testing
make test            # Run all tests
make test-coverage   # Run tests with coverage report
make bench           # Run benchmarks

# Docker
make docker-build    # Build Docker image
make docker-run       # Run Docker container
make docker-compose-up # Start all services
make docker-compose-down # Stop all services

# Database
make migrate-up      # Run database migrations
make migrate-down    # Rollback migrations

# Documentation
make swagger          # Generate Swagger docs
make docs              # Open local documentation

# Code Quality
make fmt               # Format code
make lint              # Run linter
make vet               # Run go vet
make check             # Run all checks (fmt + vet + lint + test)

# Setup and Cleanup
make setup             # Complete dev environment setup
make clean             # Remove build files
make clean-all         # Remove build files and cache

```

⚙ Configuration

Create a `.env` file in the root directory with the following variables:

```

# Server Configuration
SERVER_PORT=4000
CLIENT_NAME=evolution

# Security
GLOBAL_API_KEY=your-secure-api-key-here

# Database (Optional)
POSTGRES_AUTH_DB=postgresql://postgres:password@localhost:5432/evogo_auth?sslmode=disable
POSTGRES_USERS_DB=postgresql://postgres:password@localhost:5432/evogo_users?sslmode=disable
DATABASE_SAVE_MESSAGES=false

# Logging
WADEBUG=DEBUG
LOGTYPE=console

# Optional Features
# API_AUDIO_CONVERTER=your-audio-converter-url

```

```
# AMQP_URL=amqp://guest:guest@localhost:5672/
# WEBHOOK_URL=https://your-webhook-url.com/webhook
```

📘 Documentation

📘 Complete Documentation

Access our comprehensive documentation with guides, tutorials and API reference:

📘 Official Documentation

The documentation includes:

- ⚡ **Getting Started:** Installation, configuration and quickstart
- 🏠 **Core Concepts:** Architecture, instances, authentication
- 📄 **API Guides:** All 79 endpoints documented with examples
- 🔧 **Advanced Features:** Events, media storage, webhooks
- 🚀 **Production Deploy:** Docker, security and scalability
- 📖 **Reference:** Environment variables, error codes, FAQ

🔧 API Documentation (Swagger)

Once the server is running, you can access the interactive Swagger documentation at:

<http://localhost:4000/swagger/index.html>

Key Endpoints

- POST /instance/create - Create a new WhatsApp instance
- GET /instance/{instanceName}/qrcode - Get QR code for pairing
- POST /message/sendText - Send text message
- POST /message/sendMedia - Send media message
- GET /instance/{instanceName}/status - Get instance status
- DELETE /instance/{instanceName} - Delete instance

🐋 Docker Deployment

Using Docker Compose

```
version: '3.8'
services:
  evolution-go:
    image: evolution-go:latest
    ports:
      - "4000:4000"
    environment:
      - SERVER_PORT=4000
      - GLOBAL_API_KEY=your-secure-api-key
    volumes:
```

```
- ./logs:/app/logs  
restart: unless-stopped
```

Environment Variables

| Variable | Description | Default |
|------------------------|------------------------|-----------|
| SERVER_PORT | Server port | 4000 |
| CLIENT_NAME | Client identifier | evolution |
| GLOBAL_API_KEY | API authentication key | Required |
| DATABASE_SAVE_MESSAGES | Enable message storage | false |
| WADEBUG | WhatsApp debug level | INFO |
| LOGTYPE | Log output type | console |

□ Testing

```
# Run all tests  
make test  
  
# Run tests with coverage  
make test-coverage  
  
# Run specific test  
go test ./pkg/instance/...
```

■ Project Structure

```
evolution-go/  
└── cmd/evolution-go/          # Application entry point  
    └── evolution-go/          # Core packages  
        └── main.go             # Entry point  
    └── pkg/                  # Core packages  
        ├── instance/           # Instance management  
        ├── message/            # Message handling  
        ├── routes/              # HTTP routes  
        ├── middleware/         # HTTP middleware  
        ├── config/              # Configuration  
        └── logger/              # Logging utilities  
    └── whatsmeow-lib/           # WhatsApp library  
        └── lib.go               # WhatsApp library implementation  
    └── Dockerfile              # Docker configuration  
    └── Makefile                # Build automation  
    └── README.md               # This file
```

□ Contributing

We welcome contributions! Please see our [Contributing Guidelines](#) for details.

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

5. Open a Pull Request

Acknowledgments

- [whatsmeow](#) by [tulir](#) - The core WhatsApp library that powers this API
- [Evolution API](#) - The original Evolution API project that inspired this Go implementation

Security

For security concerns, please email: contato@evolution-api.com

License

Evolution Go API is licensed under the Apache License 2.0, with the following additional conditions:

1. **LOGO and copyright information:** In the process of using Evolution Go API's frontend components, you may not remove or modify the LOGO or copyright information in the Evolution console or applications. This restriction is inapplicable to uses of Evolution Go API that do not involve its frontend components.
2. **Usage Notification Requirement:** If Evolution Go API is used as part of any project, including closed-source systems (e.g., proprietary software), the user is required to display a clear notification within the system that Evolution Go API is being utilized. This notification should be visible to system administrators and accessible from the system's documentation or settings page. Failure to comply with this requirement may result in the necessity for a commercial license, as determined by the producer.

Please contact contato@evolution-api.com to inquire about licensing matters.

Apart from the specific conditions mentioned above, all other rights and restrictions follow the Apache License 2.0. Detailed information about the Apache License 2.0 can be found at <http://www.apache.org/licenses/LICENSE-2.0>.

Community & Support

- [WhatsApp Group](#): Join our community for support and discussions
- [Discord Community](#): Real-time chat with developers and users
- [GitHub Issues](#): Report bugs and technical issues
- [Documentation](#): Official documentation

Support the Project

If you find Evolution Go API useful, please consider supporting the project:

- Star the repository
- Report bugs and issues
- Suggest new features
- Contribute code
- Spread the word

Telemetry Notice

To continuously improve our services, we have implemented telemetry that collects data on the routes used, the most accessed routes, and the version of the API in use. We would like to assure you that no sensitive or personal data is collected during this process. The telemetry helps us identify improvements and provide a better experience for users.
