Dify技術仕様書

1. はじめに

Difyは、AIアプリケーション開発を簡素化するために設計されたオープンソースプラットフォームです。Backend-as-a-ServiceとLLMOpsを組み合わせることで、生成AIソリューションの開発を効率化し、開発者と非技術者の両方がアクセスできるようにします。Difyは、主要なLLMのサポート、直感的なプロンプトオーケストレーションインターフェース、高品質なRAGエンジン、柔軟なAIエージェントフレームワーク、直感的なローコードワークフロー、使いやすいインターフェースとAPIを統合しています。

本技術仕様書は、Difyプラットフォームの技術的な側面を詳細に記述することを目的としています。アーキテクチャ設計、API仕様、開発ガイド、デプロイメントと運用、およびベストプラクティスについて説明します。このドキュメントは、Difyを理解し、効果的に利用し、貢献するための包括的なリファレンスとなることを目指しています。

2. アーキテクチャ設計

2.1. 全体アーキテクチャ

Difyは、モジュール化されたアーキテクチャを採用しており、各モジュールが独立して機能するように設計されています。これにより、柔軟性とスケーラビリティが向上しています。主要なコンポーネントは、API、Worker、Webの3つのコアサービスと、Weaviate、DB (PostgreSQL)、Redis、Nginx、SSRF Proxy、Sandboxなどの依存コンポーネントで構成されています。

2.2. 主要コンポーネント

- **API**: DifyのバックエンドAPIサービスを提供し、フロントエンドや外部アプリケーションからのリクエストを処理します。
- **Worker**: 非同期タスクやバックグラウンド処理を担当し、LLMの推論やデータ処理などを実行します。

- **Web**: DifyのWebインターフェースを提供し、ユーザーがAIアプリケーションを構築・管理するためのGUIを提供します。
- **DB (PostgreSQL)**: アプリケーションのデータ、ユーザー情報、プロンプト、ログなどを保存する主要なデータベースです。
- **Redis**: キャッシュやセッション管理、タスクキューなどに使用されます。
- **Weaviate**: ベクトルデータベースとして、RAG (Retrieval Augmented Generation) 機能における知識ベースのベクトルインデックスを管理します。
- **Nginx**: リバースプロキシとして機能し、APIおよびWebサービスへのトラフィックをルーティングします。
- **SSRF Proxy**: サーバーサイドリクエストフォージェリ (SSRF) 攻撃を防ぐためのプロキシサービスです。
- **Sandbox**: コード実行などの隔離された環境を提供します。

2.3. 技術スタック

DifyのバックエンドはPython/Flask/PostgreSQLで構築されており、フロントエンドはNext.jsを使用しています。LLM推論エンジンにはDify Runtimeが採用されており、LangChainはv0.4以降削除されています。これにより、Difyはより最適化されたLLM操作を実現しています。

3. API仕様

Difyは、RESTful APIを提供しており、ほとんどの機能がAPIを通じて利用可能です。これにより、外部アプリケーションやサービスとの統合が容易になります。APIアクセスには、APIキー認証が必要です。

3.1. 主要APIエンドポイント

DifyのAPIは、チャット、ファイル、フィードバック、会話、TTS、アプリケーション、アノテーション、チャットフローなど、さまざまな機能を提供します。主要なエンドポイントの例としては、チャットメッセージの送信 (`POST /chat-messages`) などがあります。

3.2. リクエスト/レスポンスフォーマット

APIのリクエストおよびレスポンスは、通常JSON形式です。例えば、チャットメッセージの送信APIでは、ユーザー入力、ユーザー識別子、入力変数、応答モード、会話ID、ファイルリストなどがリクエストボディに含まれます。レスポンスには、イベントタイプ、タスクID、メッセージID、会話ID、モード、応答内容、メタデータ、作成タイムスタンプなどが含まれます。

3.3. 認証

すべてのAPIリクエストには、`Authorization` HTTPヘッダーにAPIキーを含める必要があります。APIキーは`Bearer {API_KEY}`の形式でプレフィックスを付けて指定します。セキュリティ上の理由から、APIキーはサーバー側に保存し、クライアント側で共有または保存しないことを強く推奨します。

4. 開発ガイド

Difyは、AIアプリケーションの迅速な開発を可能にするための様々な機能とツールを提供します。

4.1. 環境構築

Difyをローカル環境で開発する場合、Docker Composeを使用するのが最も簡単な方法です。前提条件として、CPUが2コア以上、RAMが4GB以上必要です。Docker Desktop (macOS/Windows) またはDockerとDocker Compose (Linux) をインストールした後、Difyのソースコードをクローンし、`docker/`ディレクトリで`.env.example`を`.env`にコピーして設定し、`docker compose up -d`コマンドでコンテナを起動します。

4.2. アプリケーション開発

Difyは、テキスト生成、チャットボット、エージェント、ワークフロー、チャットフローといった組み込みのアプリケーションタイプをサポートしています。直感的なプロンプトオーケストレーションインターフェースを提供し、プロンプトの変更と効果のプレビューを同じ場所で行うことができます。また、LLM、知識検索、コード、HTTPリクエスト、ツールなどのノードをサポートする視覚的なワークフローオーケストレーションインターフェースも備えています。

4.3. プラグイン開発

Difyは柔軟なプラグインシステムを提供しており、モジュールを分離することで独立した操作と外部統合を可能にします。開発者はOpenAIプラグイン標準ツールを呼び出したり、OpenAPI Specification APIをツールとして直接ロードしたりすることができます。これにより、Difyの機能を拡張し、特定のニーズに合わせたカスタムソリューションを構築することが可能です。

5. デプロイメントと運用

Difyは、様々な環境でのデプロイメントをサポートしており、柔軟な運用が可能です。

5.1. デプロイメント方法 (Docker Compose)

Difyのセルフホストデプロイメントには、主にDocker Composeとローカルソースコードからの起動の2つの方法があります。Docker Composeを使用する場合、DifyのGitHubリポジトリをクローンし、`docker`ディレクトリ内の`.env.example`ファイルを`.env`としてコピーし、必要に応じて環境変数を設定した後、`docker compose up -d`コマンドでコンテナを起動します。これにより、API、Worker、Web、データベース、Redis、Weaviateなどのすべてのサービスが起動します。

5.2. システム要件

Difyをデプロイするための最小システム要件は、CPUが2コア以上、RAMが4GB以上です。Docker Desktopを使用する場合は、Docker VMに最低2つのvCPUと8GBの初期メモリを割り当てることを推奨します。Linux環境では、Docker 19.03以降とDocker Compose 1.28以降が必要です。

5.3. アップグレード

Difyをアップグレードするには、`dify/docker`ディレクトリに移動し、`docker compose down`で既存のサービスを停止し、`git pull origin main`で最新のソースコードを取得します。その後、`docker compose pull`で最新のイメージをプルし、`docker compose up -d`でサービスを再起動します。`.env.example`ファイルが更新されている場合は、ローカルの`.env`ファイルもそれに合わせて更新する必要があります。

6. ベストプラクティス

Difyを効果的かつ安全に運用するためのベストプラクティスを以下に示します。

6.1. セキュリティ

- **APIキーの管理**: APIキーは機密情報であり、サーバーサイドで安全に保管し、クライアントサイドには公開しないでください。定期的にAPIキーをローテーションすることを検討してください。
- **アクセス制御**: Difyの管理インターフェースへのアクセスは、信頼できるユーザーに限定し、強力なパスワードポリシーを適用してください。
- **ネットワークセキュリティ**: Difyインスタンスへのアクセスを制限するために、ファイアウォールやセキュリティグループを設定し、不要なポートを閉じます。
- **SSRF対策**: Difyに組み込まれているSSRFプロキシを活用し、サーバーサイドリクエストフォージェリ攻撃から保護します。

6.2. パフォーマンス

- **リソースの最適化**: Difyのデプロイメントには、推奨されるCPUとRAMの要件を満たす十分なリソースを割り当ててください。特に、LLMの推論やRAG処理には多くの計算リソースが必要です。
- **データベースの最適化**: PostgreSQLデータベースのパフォーマンスを定期的に監視し、必要に応じてインデックスの追加やクエリの最適化を行います。
- **ベクトルデータベースの選定**: RAG機能のパフォーマンスは、使用するベクトルデータベースに大きく依存します。Qdrantなどの推奨されるベクトルデータベースを使用し、適切なインデックス戦略を適用してください。
- **スケーラビリティ**: 高負荷が予想される場合は、DifyのAPI、Worker、Webサービスをスケールアウトすることを検討してください。

6.3. 監視とロギング

- **ログの収集と分析**: Difyはログをサポートしており、アプリケーションの動作状況を把握するためにログを収集し、分析することが重要です。異常な動作やエラーを早期に検出するために、集中ログ管理システムを導入することを検討してください。

- **メトリクスの監視**: CPU使用率、メモリ使用率、ネットワークトラフィック、API応答時間などのシステムメトリクスを監視し、パフォーマンスのボトルネックを特定します。
- **アノテーションとフィードバック**: Difyのログに基づくアノテーション機能や、人間によるフィードバックを活用して、AIアプリケーションの品質を継続的に改善します。

7. 付録

7.1. 参考資料

- Dify公式ドキュメント
- Dify GitHubリポジトリ
- Difyブログ

7.2. 用語集

- **LLM**: Large Language Model（大規模言語モデル）
- **LLMOps**: Large Language Model Operations（大規模言語モデル運用）
- **RAG**: Retrieval Augmented Generation（検索拡張生成）
- **API**: Application Programming Interface（アプリケーションプログラミングインターフェース）
- **Docker Compose**: 複数のDockerコンテナを定義し実行するためのツール
- **PostgreSQL**: オブジェクトリレーショナルデータベース管理システム
- **Redis**: インメモリデータ構造ストア
- **Weaviate**: ベクトル検索エンジン
- **Nginx**: 高性能なHTTPおよびリバースプロキシサーバー
- **SSRF**: Server-Side Request Forgery（サーバーサイドリクエストフォージェリ）