**Guide to Setting Up and Using Dify Platform**

**1. Introduction to Dify**

Dify is an open-source platform for building AI applications that combines Backend-as-a-Service (BaaS) and LLMOps to streamline the development of generative AI solutions. The name "Dify" derives from "Define + Modify," reflecting its core philosophy of defining and continuously improving AI applications.

The platform provides a comprehensive production-ready solution rather than just a collection of tools. Think of Dify as a well-designed scaffolding system that enables both developers and non-technical innovators to create sophisticated AI applications without dealing with unnecessary complexity.

**Core Capabilities**

Dify integrates several powerful features into a unified platform. The platform supports mainstream Large Language Models (LLMs) with an intuitive prompt orchestration interface that simplifies the process of designing AI interactions. It includes high-quality Retrieval-Augmented Generation (RAG) engines that can extract text from PDFs, PowerPoint presentations, and other common document formats. The flexible AI Agent framework allows you to define agents based on LLM Function Calling or ReAct patterns, with access to over 50 built-in tools including Google Search, DALL·E, Stable Diffusion, and WolframAlpha.

The platform features an intuitive low-code workflow builder that enables visual application design with drag-and-drop functionality. All capabilities are accessible through easy-to-use interfaces and RESTful APIs, allowing seamless integration into existing business logic. Dify also provides robust LLMOps features for monitoring and analyzing application logs and performance over time, enabling continuous improvement based on production data.

**Application Types**

Dify supports two primary application types tailored to different use cases. Chatflow applications are designed for conversational scenarios, including customer service, semantic search, and other applications that require multi-step logic in response construction. These applications support conversation history (Memory), annotated replies, and enhanced question understanding capabilities.

Workflow applications are geared towards automation and batch processing scenarios, making them suitable for high-quality translation, data analysis, content generation, and email automation tasks. These workflows reduce system complexity by breaking down complex tasks into smaller steps, reducing reliance on prompt engineering while enhancing performance and stability.

**2. Why Choose Dify: Key Advantages**

**Open Source and Community-Driven**

Dify is co-created by a dedicated professional team and a vibrant community of over 180,000 developers worldwide. The project has garnered more than 114,000 stars on GitHub, demonstrating its widespread adoption and active development. This collaborative approach ensures rapid iteration, robust features, and continuous improvement driven by real-world usage feedback.

Being open-source provides transparency, flexibility, and freedom from vendor lock-in. Organizations can inspect the code, contribute improvements, and deploy the platform according to their specific requirements without being constrained by proprietary limitations.

**Educational and Enterprise Accessibility**

For educational institutions and students, Dify Cloud offers free access with a Sandbox plan that includes 200 free OpenAI calls, requiring no credit card for sign-up. This makes it an ideal platform for learning prompt engineering, exploring agent technologies, and building first AI applications. Over 60,000 developers built their first AI application on Dify even before GPTs were introduced.

For enterprises, Dify provides deployment flexibility with options for cloud-hosted services or self-hosted installations on private infrastructure. Banks and technology companies are deploying Dify as an internal LLM gateway, facilitating generative AI adoption with centralized governance while maintaining full control over sensitive data.

**Comprehensive Feature Set**

Unlike many AI development tools that offer individual components, Dify provides a complete production-ready solution. The platform enables rapid prototyping and iteration, allowing startups to build MVPs, secure funding, and win customer contracts efficiently. Established businesses can enhance existing applications with LLM capabilities by using Dify's RESTful APIs to separate prompts from business logic while tracking data, costs, and usage through the management interface.

The platform's visual interface eliminates the need for extensive coding skills, making AI application development accessible to both technical and non-technical team members. Real-time updates reflect changes across all clients, enabling collaborative development and rapid iteration cycles.

**Security and Data Control**

Dify prioritizes data security with encryption at rest for API keys and other sensitive information. When using Dify Cloud, data is securely stored on AWS servers in the US-East region with the highest standards of cloud storage solutions. For organizations requiring complete data sovereignty, self-hosted deployment options ensure that all data remains within controlled infrastructure.

Application data is anonymized in Dify Cloud to ensure privacy while reducing encryption and decryption overheads. The platform provides flexible security options that allow organizations to choose the deployment model that best meets their compliance and security requirements.

**3. Setting Up Dify Cloud (Online Hosted Version)**

**Prerequisites**

To use Dify Cloud, you will need a GitHub or Google account for authentication. Additionally, you will need an OpenAI API key or access to other supported LLM providers to power your applications. The Sandbox plan provides 200 free OpenAI calls to help you get started without immediate costs.

**Registration and Initial Setup**

Begin by navigating to the Dify Cloud platform at https://cloud.dify.ai/apps. Sign up using your GitHub or Google account through the provided authentication options. The platform uses Multi-Factor Authentication (MFA) by default for accounts created with email addresses, sending a verification code to your email each time you log in.

After successful authentication, you will either create a new Workspace or join an existing one. Workspaces serve as collaborative environments where team members can work together on AI applications. Choose an appropriate workspace name that reflects your organization or project.

**Configuring Model Providers**

Once your workspace is established, navigate to the model configuration section. You can choose to configure your own model provider by entering your API keys, or use Dify's hosted model provider for quick setup. Supported providers include OpenAI, Anthropic, Azure OpenAI, and various open-source models.

To add a model provider, access the settings menu, select model providers, and click to add a new provider. Enter your API key securely, and the platform will validate the connection. Remember that API keys are encrypted at rest, ensuring that only you have access to your credentials.

**Creating Your First Application**

With your model provider configured, you can now create your first application. Click on the "Create Application" button and choose between Chatflow for conversational applications or Workflow for automation tasks. Select a template to start quickly, or begin with a blank canvas for complete customization.

The visual workflow builder allows you to add nodes by clicking the "+" button and selecting from available components such as LLM nodes, knowledge retrieval nodes, conditional branches, HTTP requests, and tools. Configure each node by clicking on it and setting the required parameters. Use variables to pass data between nodes and create dynamic application logic.

Test your application using the Preview button, which opens a testing interface where you can interact with your application in real-time. Monitor the execution flow and debug any issues before publishing.

**Publishing and Accessing Your Application**

When your application is ready for production, click the Publish button to make it available. Published applications can be accessed through embedded web widgets, shared links, or API endpoints. Navigate to the API Access section in the left-side menu to view your application's API credentials and documentation.

**4. Setting Up Dify Community Edition (Self-Hosted)**

**System Requirements**

Before installing Dify Community Edition, ensure your machine meets the minimum system requirements. You will need a server or local machine with sufficient CPU, RAM, and storage capacity. Docker and Docker Compose are required for the recommended installation method.

For production deployments, consider specifications that can handle your expected user load and data processing requirements. The platform can be deployed on various cloud providers including AWS, Google Cloud, Azure, or on-premises infrastructure.

**Installing Docker and Docker Compose**

The easiest way to deploy Dify Community Edition is through Docker Compose. Begin by installing Docker Desktop according to your operating system. For Windows users, download Docker Desktop for Windows from the official Docker website. Mac users should download Docker Desktop for Mac. Linux users can follow the Docker Compose installation guide available in the Docker documentation.

Download the appropriate installation package for your operating system and proceed with the installation. Once installed, open Docker Desktop and follow the default settings to complete the setup. Avoid modifying advanced settings unless you have specific requirements and understand the implications.

**Deploying Dify Community Edition**

Open your terminal (Terminal on Mac, CMD or PowerShell on Windows) and execute the following command to clone the Dify repository:

git clone https://github.com/langgenius/dify.git

cd dify/docker

Navigate to the docker directory under the Dify main directory. Before starting the services, review the .env.example file to understand available configuration options. Copy this file to .env and customize values according to your requirements, such as database connections, Redis configuration, and security settings.

Execute the one-click startup command:

docker-compose up -d

This command will start all required containers including the web interface, database, Redis cache, vector database (Weaviate), API server, and worker processes. The initial startup may take several minutes as Docker downloads the necessary images and initializes the services.

Monitor the startup process by checking container status with docker-compose ps. All containers should show as "running" or "healthy" when the deployment is successful.

**Initial Configuration**

Once all containers are running, open your web browser and navigate to http://localhost/install (or your server's IP address if deploying remotely). This will present the Dify installation page where you can set up your admin account.

Create your administrative credentials by providing an email address and secure password. This account will have full access to manage the Dify platform, including creating applications, managing users, and configuring system settings.

After creating the admin account, log in to the main Dify interface. The self-hosted version provides the same functionality as Dify Cloud but runs entirely within your controlled infrastructure.

**Configuring Model Providers**

Navigate to Settings and select Model Providers to add your LLM API keys. You can add multiple providers and switch between them as needed. Enter your API credentials securely, and test the connection to ensure proper configuration.

For organizations using private or self-hosted LLMs, configure the custom model endpoints in the settings. This allows integration with models deployed within your infrastructure or those from specialized providers.

**Ongoing Maintenance**

Regularly update your Dify installation to benefit from new features, security patches, and performance improvements. To update, pull the latest changes from the repository and restart the containers:

cd dify

git pull

cd docker

docker-compose down

docker-compose up -d

Configure automated backups for the PostgreSQL database and any uploaded files to prevent data loss. Use Docker volumes or external backup solutions to maintain regular snapshots of your data.

Monitor system logs through Docker commands or by integrating with logging platforms like Grafana. The platform provides metrics at the application, tenant, and message level for comprehensive monitoring.

**5. Using Dify API with Your Workflow**

**Understanding API Authentication**

All Dify API requests require authentication using an API Key. Each application you create in Dify has its own unique API key that grants access to that specific application's endpoints. These keys should be stored securely on your server-side code and never exposed in client-side applications to prevent security breaches.

To obtain your API key, navigate to your application in the Dify interface and click on "API Access" in the left-side menu. Here you will find your API key along with comprehensive documentation for available endpoints.

**API Key Configuration**

Based on your uploaded Postman collection, your API requests are configured to use bearer token authentication. Set up your API key as an environment variable in your API client or application. In Postman, configure the Auth-ChatGPT-Text variable with your actual API key.

For workflow applications, the API key enables access to workflow execution endpoints, log retrieval, and status monitoring. Different applications will have different API keys, allowing you to segregate access and track usage per application.

**Executing Workflows via API**

The primary endpoint for running workflows is POST /v1/workflows/run. This endpoint executes your predefined workflow with specified inputs and returns results based on the configured response mode.

**Request Structure**

Your workflow execution request should include the following components:

**inputs**: A key-value object containing all workflow variables. Each key corresponds to a variable name defined in your workflow. For text inputs, provide the string value directly. For file inputs (images, audio, video), you must first upload the file using the file upload endpoint and then reference it using the upload\_file\_id.

**response\_mode**: Choose between "blocking" for synchronous responses or "streaming" for incremental updates. Blocking mode waits for the complete workflow execution before returning results, with a Cloudflare timeout limit of 100 seconds. Streaming mode returns events in real-time as the workflow progresses, which is recommended for long-running workflows that exceed 15 minutes.

**user**: A user identifier string that helps track usage and conversations per user. This can be any unique identifier such as an email address or user ID from your system.

Example request for text input:

{

"inputs": {

"InputText": "Your text content here"

},

"response\_mode": "blocking",

"user": "user@example.com"

}

**Handling File Uploads**

When your workflow requires file inputs (images, voice recordings, or video files), follow a two-step process. First, upload the file using the POST /v1/files/upload endpoint. This endpoint accepts multipart form data with the file and user identifier.

The upload endpoint returns a response containing an upload\_file\_id. Use this identifier in your workflow execution request to reference the uploaded file.

Structure your workflow execution request with file input as follows:

{

"inputs": {

"InputVoice": {

"transfer\_method": "local\_file",

"upload\_file\_id": "your-upload-file-id-here",

"type": "audio"

}

},

"response\_mode": "blocking",

"user": "user@example.com"

}

The type field should match your file type: "audio" for voice files, "image" for images, or "video" for video files.

**Working with Response Modes**

**Blocking Mode Response**

In blocking mode, the API waits for the workflow to complete and returns a comprehensive response containing the workflow run ID, task ID, execution status, outputs, token usage, and timing information. The outputs object contains all results from your workflow's end nodes.

Parse the response to extract the data you need from the outputs object. The status field indicates whether the workflow succeeded or encountered errors.

**Streaming Mode Response**

Streaming mode returns Server-Sent Events (SSE) as the workflow executes. You will receive multiple event types including workflow\_started, node\_started, node\_finished, and workflow\_finished. Each event contains relevant data about the execution progress.

Implement an SSE client in your application to handle these events. Listen for workflow\_finished to know when the execution completes and to retrieve final outputs. For long-running workflows, streaming mode prevents timeout issues and provides real-time feedback.

**Retrieving Workflow Information**

Use GET /v1/info to retrieve metadata about your workflow application, including available input variables and their types. This endpoint helps you understand what parameters your workflow accepts.

To get detailed information about a specific workflow execution, use GET /v1/workflows/run/{workflow\_run\_id}. This endpoint returns the current status, inputs, outputs, elapsed time, token usage, and any errors that occurred. Use this for polling the status of long-running workflows initiated in streaming mode.

**Accessing Workflow Logs**

The GET /v1/workflows/logs endpoint retrieves historical execution logs for your workflow. This is valuable for monitoring, debugging, and analyzing workflow performance over time. Filter logs by date range, status, or user to find specific executions.

**Error Handling**

Implement robust error handling in your API integration. Common response codes include:

* 200: Successful execution
* 400: Bad request (invalid parameters or missing required fields)
* 401: Unauthorized (invalid or missing API key)
* 404: Resource not found (invalid workflow ID or run ID)
* 500: Server error (workflow execution failure)

Parse error responses to provide meaningful feedback. The error field in the response contains details about what went wrong during execution.

**Rate Limiting and Best Practices**

Store API keys securely using environment variables or secret management systems. Never hardcode keys in your application code or commit them to version control. Implement retry logic with exponential backoff for transient failures. Monitor your API usage to stay within rate limits and avoid unexpected costs.

For long-running workflows exceeding 15 minutes, always use streaming mode or implement asynchronous polling patterns. Cache workflow results when appropriate to reduce unnecessary API calls. Log all API interactions for debugging and audit purposes.

**Integration Example**

Your Postman collection demonstrates a typical workflow integration pattern. First, check application parameters using the parameters endpoint to understand required inputs. Upload any files needed for the workflow. Execute the workflow with appropriate inputs and response mode. Monitor the execution status and retrieve results. Access logs for historical analysis and debugging.

This pattern ensures reliable integration with comprehensive error handling and efficient resource usage.

**6. Best Practices and Tips**

**Workflow Design**

Design workflows with modularity in mind, breaking complex tasks into discrete nodes that can be tested and modified independently. Use conditional branching to handle different scenarios and edge cases gracefully. Leverage variables to maintain state across nodes and create dynamic application logic.

Test workflows thoroughly using the preview feature before publishing to production. Use the step-by-step execution mode to identify and fix issues in individual nodes. Document your workflow logic and variable purposes for team collaboration.

**Security Considerations**

Always use HTTPS endpoints for API communication to protect data in transit. Rotate API keys periodically and immediately revoke compromised keys. Implement proper authentication and authorization in your applications that consume Dify APIs. For self-hosted deployments, secure your server infrastructure with firewalls, regular security updates, and access controls.

**Performance Optimization**

Monitor token usage to optimize costs and performance. Use streaming mode for workflows with long execution times to improve user experience. Implement caching strategies for frequently requested data. Configure appropriate timeout values based on your workflow complexity.

For high-traffic applications, consider deploying multiple Dify instances with load balancing. Use the LLMOps features to analyze performance metrics and identify optimization opportunities.

**Collaboration and Version Control**

Take advantage of Dify's collaboration features to work with team members on application development. Use version control to track changes to workflows and rollback if needed. Document configuration changes and maintain clear communication within your team about application modifications.

**Continuous Improvement**

Leverage the monitoring and analytics capabilities to understand how your applications are performing in production. Collect user feedback and iterate on prompts, knowledge bases, and workflow logic. Stay updated with Dify's release notes to benefit from new features and improvements.

Participate in the Dify community through GitHub discussions, Discord channels, and community forums. Share your experiences and learn from other developers' implementations.