

Dify EE 101B

# 技術基礎とデプロイ 演習

Dify エコシステムパートナー トレーニングシリーズ

Japan FDE Team

Infrastructure for Intuitive LLM App Development

↓ PDF

# コースの目標



## コアアーキテクチャ

Dify EE のコアサービス構成、トラフィック経路、実行モデルを理解する。



## 最小デプロイ

K8s/Helm デプロイスキルを習得し、「最小限の使用可能な環境」を独自に構築できるようにする。



## 技術サービス

基本的なセキュリティと可観測性の知識を備え、トラブルシューティングと技術サポートの方法を習得する。



# コース概要

- 1. [アーキテクチャとモジュール] コアサービスの分解と通信リンク
- 2. [データ構造] データベーススキーマとビジネスの対応付け
- 3. [デプロイ実習] Hands-on Labs (K8s/Helm)
- 4. [技術サービス] トラブルシューティングガイドとベストプラクティス



# 第1章：アーキテクチャとモジュールの概要



# 1.1 Dify エディションとデプロイ方法

| シナリオが形態を決定する

Edition	Deployment	Capability	Scene	Availability & Reliability
 <b>Dify EE</b>	K8s / Helm	<b>Enterprise Full</b>	Production / High Scale	 <b>High Availability (HA)</b> Auto-scaling, Rolling Updates, Fault Tolerance
 <b>Dify EE</b>	Docker Compose	<b>Enterprise Full</b>	PoC / Test / SMB	 <b>Single Node</b> Manual Failover, Downtime on Update
 <b>Dify CE</b>	Docker Compose	<b>Community</b>	Dev / Personal	 <b>Basic</b> No SLA, Best Effort
 <b>Premium</b>	AWS / GCP Image	<b>CE + Branding</b>	Cloud Market Launch	 <b>VMSLA</b> Cloud Provider VM Guarantee

 Production Ready     Use with Caution

 PDF

# 1.2 Dify EE アーキテクチャ総覧

## マイクロサービスアーキテクチャとトラフィックエントリ

- クリティカルパス
  - 1. トラフィックエントリ: Ingress -> Gateway (Nginx/Caddy)
  - 2. コアアプリ: API / Web / Enterprise がビジネスロジックを処理
  - 3. 非同期処理: Redis Queue -> Worker がタスクを消費
  - 4. 永続化: Postgres (ビジネスデータ) + Redis (キャッシュ) + VectorDB
  - 5. 外部依存: S3 (ファイル) + LLM Providers



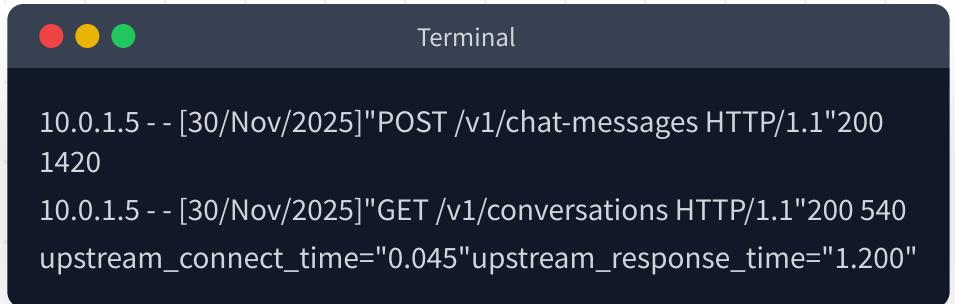
# 1.3 サービス分類 (Service Taxonomy)

- **1. エントリ層:** Gateway (統一リバースプロキシ)
- **2. アプリ層:** API, Web, Enterprise, Enterprise-Frontend
- **3. 非同期層:** Worker (タスク実行), Worker Beat (タスクスケジューリング)
- **4. 監査層:** Enterprise Audit (ログ記録)
- **5. 補助層:** Sandbox (コードサンドボックス), SSRF Proxy, Unstructured
- **6. プラグイン層:** Plugin Daemon/Manager/Connector/Controller



# 1.4 コアサービス/Gateway & API

- Gateway (エントリゲートウェイ)
  - Caddy/Nginx ベース、統一ルーティング
  - ポート: 80 (HTTP) / 443 (HTTPS)
  - 役割: リバースプロキシ、CORS、静的リソースキャッシュ
- API Service (コアバックエンド)
  - Python/Flask アプリ、ビジネスロジックの中核
  - ポート: 5001
  - 役割: 認証、モデル呼び出し、ワークフローオーケストレーション
  - 依存: Postgres, Redis, VectorDB



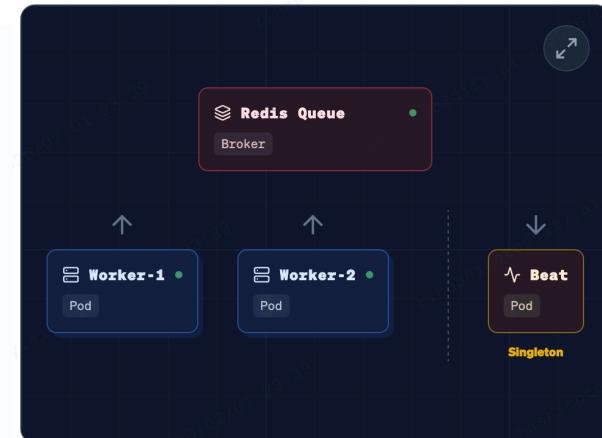
A screenshot of a terminal window titled "Terminal". The window shows two log entries from an application server. The first entry is a POST request for "/v1/chat-messages" with a status code of 200 and a duration of 1420 microseconds. The second entry is a GET request for "/v1/conversations" with a status code of 200 and a duration of 540 microseconds. Both entries include metrics for upstream connect and response times.

```
10.0.1.5 -- [30/Nov/2025]"POST /v1/chat-messages HTTP/1.1"200  
1420  
10.0.1.5 -- [30/Nov/2025]"GET /v1/conversations HTTP/1.1"200 540  
upstream_connect_time="0.045"upstream_response_time="1.200"
```

# 1.5 非同期処理/Worker & Beat

## | Dify の心臓と脈拍

- Worker (タスク実行者)
  - Redis Queue から Celery タスクを消費
  - Service ポートなし、Redis へのクライアントとして動作
  - 処理: RAG インデックス、ファイル解析、メール送信、長時間ワークフロー
  - 水平スケーリング可能 (HPA)
- Worker Beat (スケジューラ)
  - 定期タスク (Crontab) を生成
  - シングルトン必須 (Replicas=1)、タスク重複防止
  - 役割: クリーンアップ、監視、同期などの周期的動作のトリガー



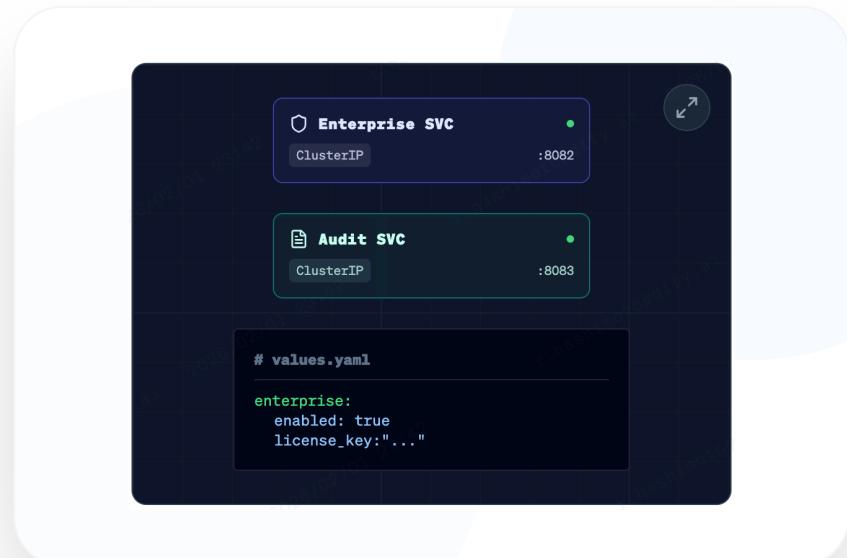
# 1.6 エンタープライズサービス (Enterprise)

- Enterprise Backend

- ポート: 8082 (HTTP) / 9000 (gRPC)
- SSO、RBAC、Team Management などのエンタープライズ機能を提供
- API Service とメインデータベースを共有

- Enterprise Audit

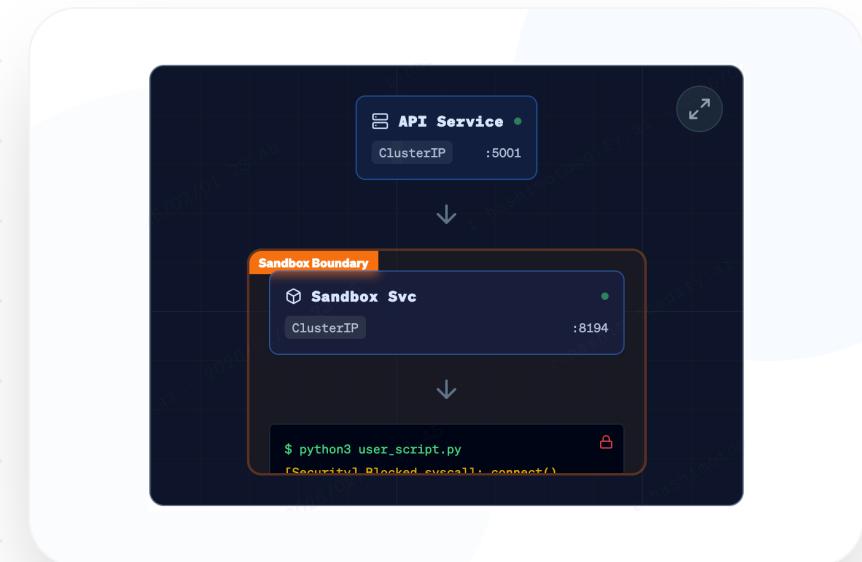
- ポート: 8083
- 独立した監査サービス、操作ログを記録
- 監査データベース (audit) に接続、コンプライアンスクリアをサポート



# 1.7.1 コードサンドボックス (Sandbox)

## | 分離された安全な実行環境

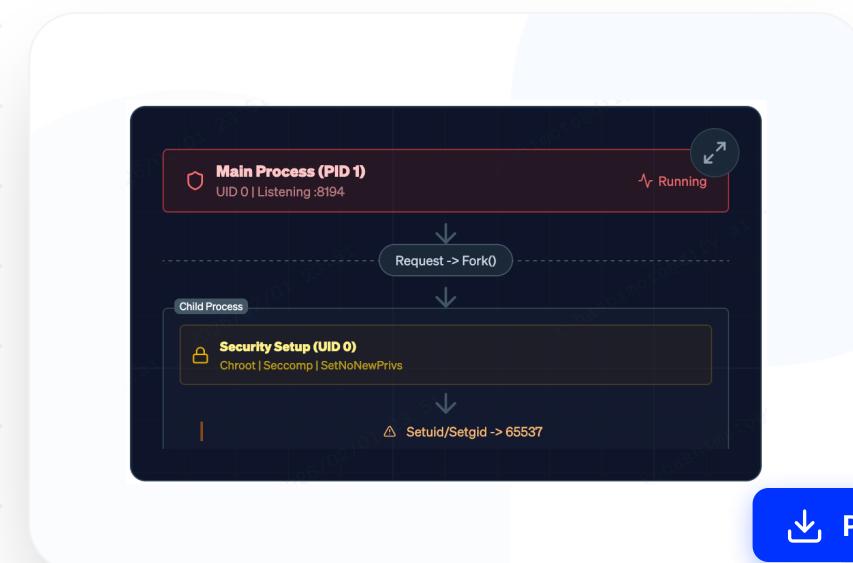
- コアメカニズム (dify-sandbox)
  - Seccomp: システムコール (Syscall) を制限、ネットワークアクセス制御を含む (非独立 NetNS)
  - Chroot: ファイルシステムアクセス範囲を制限
  - Privilege: setuid/setgid 権限降格 + SetNoNewPrives 権限昇格防止
- フロー
  - 1. API がコードを送信 -> Sandbox SVC (Port 8194)
  - 2. 子プロセス起動 -> InitSeccomp -> Chroot -> SetNoNewPrives
  - 3. 実行制限: ホワイトリスト内のシステムコールのみ許可、デフォルトでネットワークなし



# 1.7.2 なぜ Sandbox に Root が必要なのか？

## | メインプロセスリスナー & 子プロセス権限降格モデル

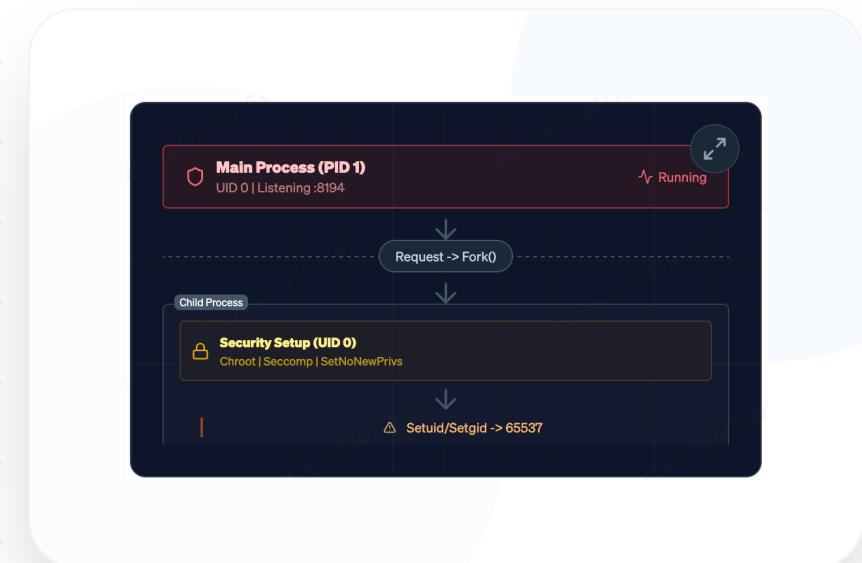
- Service Start (One-time)
  - UID 0 (Root) でメインプロセスを起動
  - Sandbox ユーザー (UID 65537) を準備、ポート 8194 をリッスン
- Request Loop (Fork Model)
  - 1. コード実行リクエスト受信 -> 子プロセスを Fork
  - 2. Main Process: リッスンを継続、Root 権限を保持
  - 3. Child Process: 権限降格とコード実行を行う
- Child Isolation
  - Chroot & Seccomp & SetNoNewPrives
  - Setuid -> 65537: 最終的にユーザー ID を切り替えてユーザコードを実行



# 1.7.3 SSRF 保護プロキシ

## | イントラネット侵入を防ぐ重要な防衛線

- SSRF リスク
  - ユーザーが API ツールでイントラネット URL (例: <http://192.168.1.1/admin>) を入力
  - サーバーが直接リクエストすると、イントラネットの機密情報が漏洩する
- 保護メカニズム
  - 1. すべての外部 HTTP リクエストを Squid Proxy (Port 3128) 経由にする
  - 2. Proxy がターゲットドメイン IP を解決
  - 3. ACL フィルタリング: [10.0.0.0/8](http://10.0.0.0/8), [172.16.0.0/12](http://172.16.0.0/12), [192.168.0.0/16](http://192.168.0.0/16), [127.0.0.1](http://127.0.0.1)などのプライベート範囲をブロック
  - 4. パブリック IP トラフィックのみ許可



# 1.7.4 その他の補助サービス



## Unstructured ETL

非構造化ドキュメント解析サービス (Port 8000)。PDF/PPT/HTMLなどの複雑なフォーマットを処理し、RAG 用にテキストを抽出



PDF

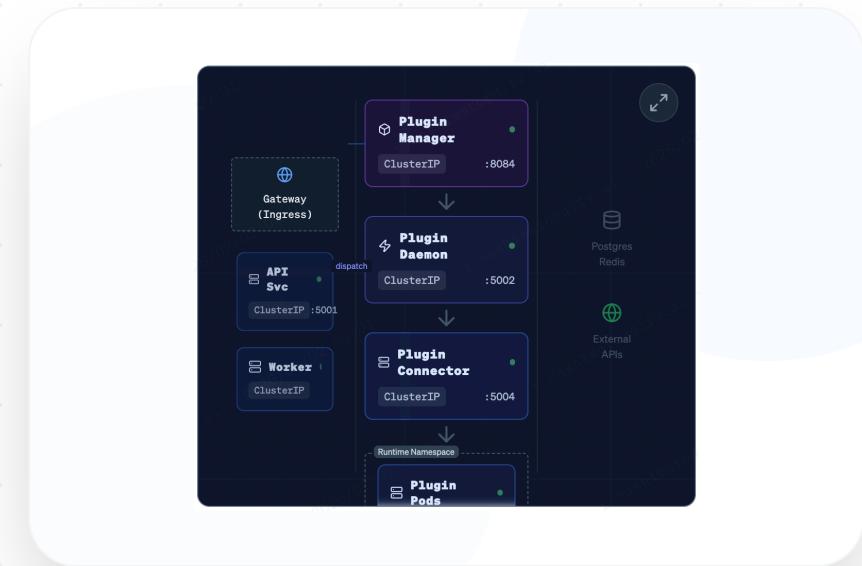
# 1.8 プラグインエコーシステムアーキテクチャ



PDF

# プラグインサービスコンポーネント

- Plugin Manager
  - プラグインマーケットプレイスバックエンド、プラグインメタデータと権限を管理
  - ポート: 8084 (HTTP) / 9084 (gRPC)
- Plugin Daemon
  - ランタイムデーモン、プラグインのライフサイクルと呼び出しを担当
  - API/Worker からの **dispatch** リクエストを受信
  - ポート: 5002 (API)
- Plugin Connector & Runtime
  - Connector: SRI プロトコル実装、K8s/Lambda で Pod を管理
  - Runtime: 実際にプラグインコードを実行する Pod、Sidecar 経由で外部と通信



# 1.9 デプロイ側の重要な注意事項

- **⚠ Beat Singleton:** `workerBeat.enabled` は1つの Pod でのみ有効にし、決して複数レプリカにしないでください
- **🔒 外部アクセス:** デフォルトは ClusterIP。外部アクセスには Ingress または LoadBalancer を構成します
- **💾 永続化:** 本番環境では、Chart 組み込みの StatefulSet ではなく、マネージド RDS/Redis/S3 を強く推奨します
- **🛡️ セキュリティ:** デフォルトのパスワードを変更し、TLS 証明書を構成し、SSRF Proxy を有効にしてください



# 第1章のまとめ

- ✓ コアリンク: Gateway -> API -> Worker は最も基本的なビジネスの三角形です
- ✓ エンタープライズ機能: Enterprise + Audit は、ビジネスから独立した管理層を提供します
- ✓ プラグインシステム: 拡張機能の動的ロードを担当する独立したマイクロサービスグループ
- ✓ 本番環境のアドバイス: 状態データ (DB) の分離とステートレスサービス (App) の拡張に焦点を当ててください

 PDF

# 第2章：Dify EE データ構造

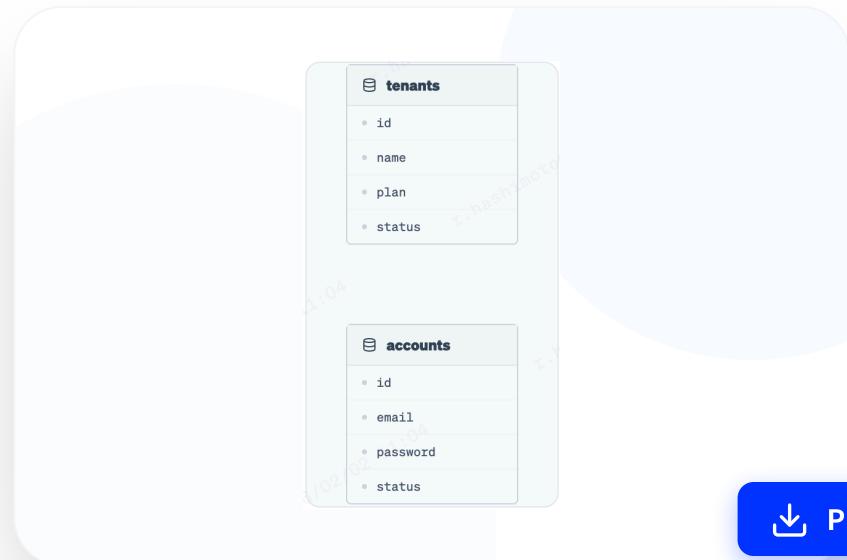


PDF

# 2.1 メインデータベース/テナントとアカウント

## | Table Group: Identity & Auth

- DB名: [dify](#)
- 最も中心的な基本データ。「誰」が「どこ」で「何」をするかを記録
  - **tenants**: テナント/ワークスペース、データ分離の境界
  - **accounts**: 登録ユーザー、テナントと多対多で関連付け
  - **tenant\_account\_joins**: 関連テーブル、ロール (Role) を定義
  - **account\_integrates**: サードパーティログイン (OAuth)

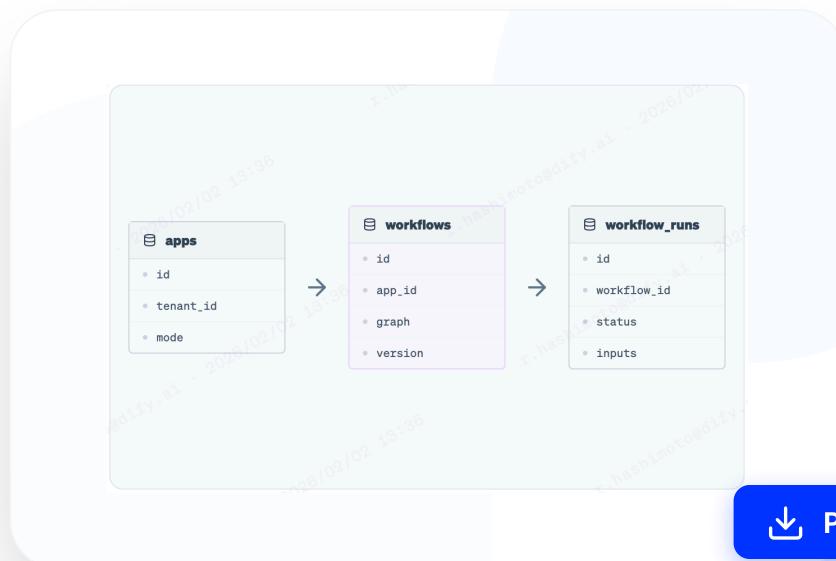


PDF

# 2.2 メインデータベース/アプリとワークフロー

## | Table Group: Application Logic

- App 定義
  - apps: アプリメタデータ (Name, Mode, Icon)
  - app\_model\_configs: アプリモデル、パラメータ、および Prompt 設定
- Workflow 定義
  - workflows: DSL バージョン管理とグラフ構造定義
  - workflow\_runs: 実行インスタンス、ステータスと所要時間 を記録
  - workflow\_node\_executions: ノードレベルの実行詳細



PDF

# 2.3 メインデータベース/ナレッジベース (RAG)

## | Table Group: Dataset & Document

- データ階層
  - 1. datasets: ナレッジベースコンテナ
  - 2. documents: アップロードされたファイル/データソース
  - 3. document\_segments: 分割されたテキストチャンク (Chunk)
- 主要な関連付け
  - dataset\_collection\_bindings: VectorDB Collection を関連付け
  - embeddings: Embedding 結果をキャッシュ (Hash 対応)

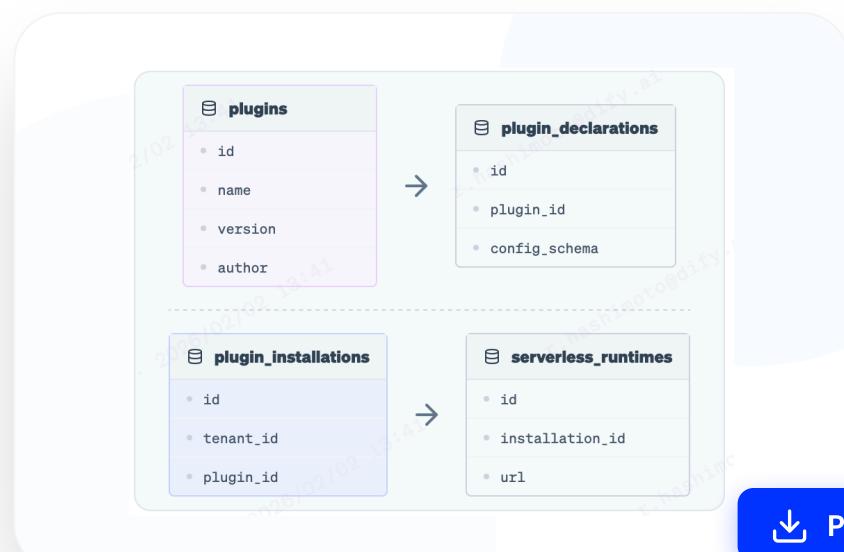


PDF

# 2.4 メインデータベース/プラグインエコシステム

## Table Group: Lifecycle & Runtime

- プラグインのライフサイクル、インストール記録、実行時ステータスを管理
- コアテーブル構造
  - `plugins`: プラグインパッケージメタデータ (Name, Tag, Author)
  - `plugin_declarations`: `manifest.yaml` 解析後の設定
  - `plugin_installations`: テナントインストール記録 (Tenant -> Plugin)
- ランタイムテーブル
  - `serverless_runtimes`: 対応する Serverless 関数のエンドポイント
  - `tool/model_installations`: 特定の機能のインスタンス化記


[PDF](#)

## 2.5 その他の重要なデータベース



### Audit DB

DB: `audit`で、`audit_logs` を保存し、すべての操作動作を記録。コンプライアンスの不变性要件を満たすために独立して保存



### Enterprise DB

DB: `enterprise`で、`sys_users` (管理者)、`licenses`、`member_groups` (RBAC) を保存

# 第2章のまとめ

- ✓ マルチDBアーキテクチャ: ビジネス(dify)、監査(audit)、プラグイン(plugin)、エンタープライズ(enterprise) の4つのDB分離
- ✓ トラブルシューティングの鍵: 問題は通常 `workflow_runs` (実行失敗) または `document_segments` (リコール失敗) にあります
- ✓ データ分離: Tenant ID は、すべてのビジネステーブルを貫通するコアフィールドです

PDF

# 第3章：ハンズオンラボ



PDF

# 3.1 ラボの前提条件



## ✓ 環境準備

Linux/macOS, Docker Desktop または Minikube/Kind



## ✓ ツールチェーン

kubectl, helm, git がインストールされ、パスが設定されていること



## ○ リソース要件

少なくとも 4 Core CPU、8GB RAM (K8s + DB 実行用) を推奨



PDF

# Lab A/ values.yaml の作成

- 目標: ローカルに適した最小構成を作成する
- コア設定
  - 1. `global.mode: api` (バックエンドのみ) または `standard` (フルスタック) を選択
  - 2. `service.type`: ローカル開発は `NodePort` 推奨、本番環境は `ClusterIP`
  - 3. `external`: ホストマシンの DB を再利用する場合、ホスト IP を正しく設定する (`localhost` 不可)
- なぜ補助ツールを使用するのか
  - 公式 Chart には 500 以上の設定項目があり、手書きは間違いややすい
  - バージョンアップグレード時に、古い設定ファイルに新しい重要なフィールドが欠けている可能性がある
  - ツールは HA (高可用性) とシングルモードのパラメータの違いを自動的に処理する



## Helm Values Generator

Dify EE 設定ジェネレーター。モジュール式設定と自動スイッチカスケード設定をサポートし、標準 YAML をエクスポート。

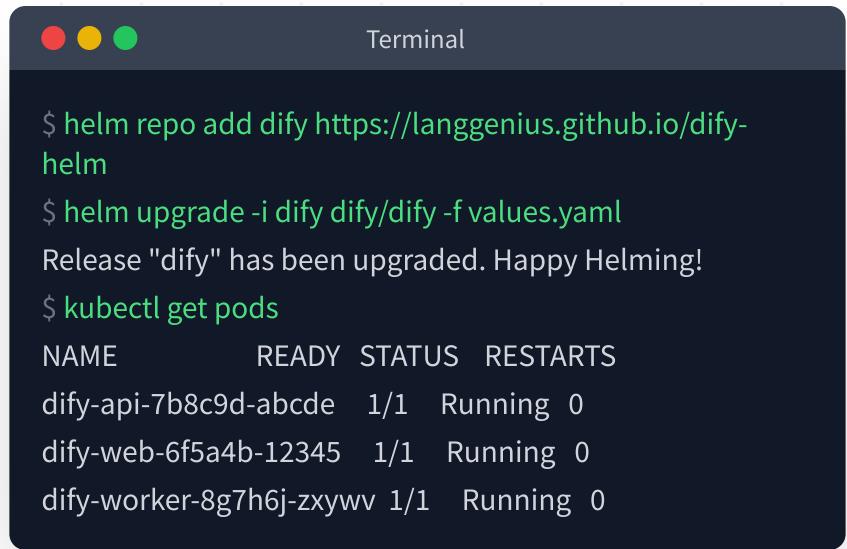


## Helm Watchdog

設定ドリフト検出ツール。「ローカル values.yaml」と「公式最新 Chart」を比較し、欠落している新しいフィールドや廃止された古いフィールドを強調表示します。

# Lab B/ Dify EE のインストール

- インストール実行
- 検証手順
  - 1. `kubectl get pods` すべての Pod が Running であることを確認
  - 2. `kubectl logs -f deploy/dify-api` エラーがないことを確認
  - 3. `http://console.dify.local` にアクセスし、ログインページを確認
- よくある問題
  - イメージプル失敗 (ImagePullBackOff) -> ネットワーク/プロキシを確認
  - DB 接続失敗 (CrashLoopBackOff) -> パスワード/Host を確認



The screenshot shows a macOS Terminal window titled "Terminal". It displays the following command and its output:

```
$ helm repo add dify https://langgenius.github.io/dify-helm
$ helm upgrade -i dify dify/dify -f values.yaml
Release "dify" has been upgraded. Happy Helming!
$ kubectl get pods
NAME           READY   STATUS    RESTARTS
dify-api-7b8c9d-abcde  1/1     Running   0
dify-web-6f5a4b-12345  1/1     Running   0
dify-worker-8g7h6j-zxywv 1/1     Running   0
```

# Lab C/D/E/ ビジネス検証

## ■ Lab C: プラグインインストール

- Plugin マーケットで [Jina](#) (Embedding) と [OpenAI](#) (LLM) をインストール
- API Key を設定し、接続をテスト

## ■ Lab D: ナレッジベース作成

- PDF ドキュメントをアップロードし、ETL とインデックス作成プロセスを観察

## ■ Lab E: Chatflow オーケストレーション

- 単純な RAG Flow を作成: [Start](#) → [Retriever](#) → [LLM](#) → [End](#)
- Debug を実行し、Trace ビューでノードの所要時間を観察



# 第3章のまとめ

- ✓ 最小ループ: Env -> Helm -> Install -> Verify
- ✓ 検証の核心: Pod ステータスが正常であることは第一歩に過ぎず、ビジネスフローがスムーズであることがゴールです
- ✓ 実践の価値: 自ら遭遇したすべてのエラーは、将来顧客の問題を解決するための経験になります

PDF

# 第4章：顧客技術サービスガイド



# 4.1 責任の合意 (SLA)

## | 階層型サービスモデル

- なぜ階層化が必要か？
  - 効率最大化: 問題の 80% は設定や使用層に属し、フロントエンドで解決すべき
  - リソース集中: 公式チームは 20% のコアコードの難題に集中
- Partner の価値
  - Partner は単なる販売チャネルではなく、技術サービスの第一防衛線です。L1/L2 問題を独自に解決できる能力は、Advanced Partner になるための重要な指標です

### L1: 使用に関する問い合わせ



"アプリの作成方法は？Prompt の効果が悪い？"

Partner

Response: ビジネスコンサルタント / デリバリーチーム

### L2: 環境運用



"K8s デプロイエラー、証明書期限切れ、ネットワーク不通"

Partner

Response: 運用エンジニア / アーキテクト

### L3: 製品の欠陥



"コアコードのバグ、セキュリティ脆弱性、論理エラー"

Dify Official

Response: Dify コア R&D チーム

PDF

# 4.2 デバッグツール/Vibe-Debugging

## | AI をトラブルシューティングに活用

- 1. コンテキスト構築: Dify コアリポジトリをローカルに一括クローンし、AI コーディングアシスタント (Cursor / Antigravity / Claude Code) にインポート
- 2. 正確なプロンプト
  - "この設定項目は具体的に何をしますか？"
  - "エラーログはどのコードセグメントに対応しますか？"
  - "リトライポリシーを変更するには？"
- Goal: 目標は AI にすべてを修正させることではなく、調査範囲を絞り込み、「コードコンテキスト」を迅速に補完することです


**コアリポジトリリスト (Context Sources)**

AI ナレッジベースとして同じディレクトリに Clone することを推奨

 <b>langgenius/dify</b>	<span style="border: 1px solid #ccc; padding: 2px;">Core App</span>
 <b>langgenius/dify-helm</b>	<span style="border: 1px solid #ccc; padding: 2px;">K8s Charts</span>
 <b>langgenius/dify-sandbox</b>	<span style="border: 1px solid #ccc; padding: 2px;">Code Sandbox</span>
 <b>langgenius/dify-plugin-daemon</b>	<span style="border: 1px solid #ccc; padding: 2px;">Plugin Runtime</span>
 <b>langgenius/dify-official-plugins</b>	<span style="border: 1px solid #ccc; padding: 2px;">Plugins</span>

[↓ PDF](#)

# 4.3 アップグレードとロールバック SOP



## 1. バックアップ (Backup)

アップグレード前に必ず DB と VectorDB を完全バックアップする



## 2. アップグレード (Upgrade)

Image Tag を変更し、helm upgrade を実行する



## 3. 検証 (Verify)

データベース Migration ログを確認し、エラーがないことを確認する



## 4. ロールバック (Rollback)

失敗した場合：DB バックアップを復元 -> Image Tag をダウングレード

# 第4章のまとめ

- ✓ 境界の明確化：L1/L2/L3 の境界を明確にし、双方のリソースを保護する
- ✓ ツールの活用：トラブルシューティングに AI を活用し、設定比較に Helm Watchdog を使用する
- ✓ セキュリティのボトムライン：データバックアップはすべての変更操作の生命線です

[PDF](#)

# THANK YOU

---

インタラクションと質疑応答

WEBSITE

<https://dify.ai>

EMAIL

[r.hashimoto@dify.ai](mailto:r.hashimoto@dify.ai)

GITHUB

[langgenius/dify](https://github.com/langgenius/dify)

