# 技術的な手数を増やす話

ndruger

# 昔話。入社して新卒の頃に感じたこと。

配属された研究開発のチームの先輩方が非常に深い知識を持っていたために、これほど深い知識を持つ人達の中で、自分は役に立つ意見を言ったり意味のある作業が出来るのかという心配があった。

# ある出来事

ある日、TCP/IPに関する問題が起きたときに、尊敬している先輩がTCP/IPを余り知らないことを知った。

# 結果

コンピューターサイエンスのすべての分野に精通している人はいないという当たり前のことを気づいた。

先輩なら知っているだろうと思い込まず、意見を言ったり行動をすることは意味があると考えるようになった。そして、自動テストなどチームで抜けていると感じた範囲を積極的に学んで進めた。
また、専門性のみで自分がIT業界でやっていくより、広い分野を学んだ方が生き残れると思った。

# その後

- インフラ、Webフロントエンド、バックエンド、MLを学んだ。
- 新たに学ぶフレームワークやライブラリ毎に、ノートを取り、類似点を比較しながら知識を整理していった。

# 現在

インフラ・Webフロントエンド・バックエンド・MLの知識があると便利なプロジェクトをしています。

# 幅広い知識を関連付ける利点

技術の関連性を理解する。

- バックエンドの言語: Python
  - PythonはTypeScriptと同じように細かく型を付けると分かりやすくミスも減らせる。
- Webフレームワーク: FastAPI
  - FastAPIでモデル・サービス・コントローラーのレイヤーをどう扱うのかは、PhoenixやRailsと比較し、その差を理解しながら、どれに寄せると綺麗に書けるか考えればいい。
- ORM: SQLModel
  - SQLModelはSQLAlchemyに型が付いたと考えるとよく、型ベースのバリデーションの思想はCromaやZodと同様の考え方が適用できる。
    - 参考: 型情報と相性のいいバリデーションライブラリ - Zod, Croma, Pydantic

技術間の共通点や差分を意識して設計すると、よりスムーズに開発が進む。

# 問題解決の多様性

問題: CircleCIでのPRごとの自動テスト実行時に、cimg/postgresのイメージが、プロジェクトで必要なPostgreSQL拡張pgvectorをサポートしていない。

対応案

1. img/postgresのDockerイメージをベースに、pgvectorを追加したカスタムイメージをDocker Hubにアップロードし、CircleCIでそれを使用する。
2. CircleCIからpgvectorに対応しているAurora DBへ接続し、それを使用する。
3. 別のCIツールであるAWS CodeBuildを利用する。
4. CircleCIのジョブ内でPostgreSQLとpgvectorをインストールし使用する。
5. pgvectorを使用するテストを特定のマークで区別し、それらのテストだけを2の方法でAuroraで実行する。

# 問題発生時のアプローチ

遭遇した問題に冷静に対応するため、可能な解決策を列挙し、最適な方法を選択しよう。

# 今日の本題: それってChatGPTが代替してくれるのでは？

# ChatGPTに質問: circleciでcimg/postgresにpgvectorが含まれていません。どうすればpgvectorを利用したCIを実行できるでしょうか？いくつかの案を上げてください

`pgvector`はPostgreSQLのベクトル拡張で、高速なベクトル検索をサポートしています。CircleCIの`cimg/postgres`イメージにはデフォルトで`pgvector`が含まれていないため、以下の方法で`pgvector`を利用したCIを実行することができます。

1. **カスタムDockerイメージの作成**:
   - `cimg/postgres`をベースにしたカスタムDockerイメージを作成します。
   - このカスタムイメージ内で`pgvector`をインストールします。
   - 作成したカスタムイメージをDocker Hubや他のコンテナレジストリにプッシュします。
   - CircleCIの設定ファイルで、このカスタムイメージを使用するように指定します。
2. **CircleCIのジョブ内でのインストール**:
   - CircleCIのジョブの実行ステップ内で、`pgvector`をインストールするコマンドを追加します。
   - これにより、ジョブが実行されるたびに`pgvector`がインストールされます。

3. **Docker Composeを使用**:
    - `docker-compose.yml`ファイルを作成し、PostgreSQLと`pgvector`を含むサービスを定義します。
    - CircleCIの設定ファイル内で、`docker-compose`コマンドを使用してサービスを起動します。
4. **CircleCI Orbsを利用**:
    - CircleCIのOrbsは再利用可能なジョブ、コマンド、およびエグゼキュータのパッケージです。
    - `pgvector`をサポートするOrbが存在する場合、それを利用することで簡単にセットアップできます。
5. **他のCIサービスの検討**:
    - CircleCI以外のCIサービスで、`pgvector`をサポートしているものがあれば、それを利用することも考えられます。

これらの方法の中から、プロジェクトの要件や予算、維持の容易さなどの要因に基づいて最適な方法を選択することができます。

私が思いつく方法のある程度はカバーしている。

# ChatGPTに質問: 先に挙げた解決方法で最も容易な物はどれでしょうか？

先に挙げた解決方法の中で、最も容易なものは「**CircleCIのジョブ内でのインストール**」です。以下にその理由を説明します：

1. **セットアップの簡易さ**:
    - ジョブの実行ステップ内で必要なツールや拡張をインストールするコマンドを追加するだけで、特別な設定やカスタムイメージの作成が不要です。
2. **カスタムイメージの管理の不要**:
    - カスタムDockerイメージを作成する場合、そのイメージの管理や更新が必要になります。しかし、ジョブ内でのインストールを選択すると、このような追加の管理作業は不要です。
3. **柔軟性**:
    - 必要に応じてインストールするツールやバージョンを簡単に変更できます。これにより、テスト環境を迅速に調整することができます。

4. **即時のフィードバック**:
    - ジョブ内でのインストールが失敗した場合、その原因をすぐに確認し、修正することができます。

ただし、この方法にはデメリットもあります。ジョブが実行されるたびに`pgvector`などのツールをインストールする必要があるため、テストの実行時間が長くなる可能性があります。しかし、セットアップの容易さと管理の簡易さを考慮すると、この方法が最も手軽であると言えます。

# 私も同じ結論。

# ライブラリの差分や共通点を見つけて頭を整理したい場合

# ChatGPTに質問: ormとしてectoとsqlalchemyとactive recordに関して、commitのタイミングに違いはある？

`Ecto`, `SQLAlchemy`, および `Active Record` の間でのコミットのタイミングや取り扱いには、いくつかの違いがあります。以下にそれぞれのORMのコミットのタイミングに関する特徴を示します：

### Ecto

- Ectoでは、`Repo.transaction/2` を使用してトランザクションを開始します。
- トランザクション内の操作がすべて成功すると、自動的にコミットされます。
- エラーや例外が発生すると、トランザクションは自動的にロールバックされます。
- 明示的に `Repo.rollback/1` を呼び出すことで、トランザクションをロールバックすることもできます。

### SQLAlchemy

- SQLAlchemyのセッションを使用してトランザクションを管理します。
- `session.commit()` を呼び出すことで、変更をデータベースにコミットします。
- エラーや例外が発生すると、`session.rollback()` を呼び出してトランザクションをロールバックする必要があります。
- `session.autocommit` プロパティを `True` に設定することで、自動コミットモードを有効にすることもできます。

### Active Record

- Active Recordでは、`ActiveRecord::Base.transaction` メソッドを使用してトランザクションを開始します。
- トランザクションブロック内の操作がすべて成功すると、自動的にコミットされます。
- エラーや例外が発生すると、トランザクションは自動的にロールバックされます。
- 明示的なコミットやロールバックのメソッドは提供されていません。

### 比較

- **自動コミット**: EctoとActive Recordは、トランザクションブロック内の操作が成功すると自動的にコミットします。一方、SQLAlchemyでは、明示的に `session.commit()` を呼び出す必要があります。
- **ロールバック**: すべてのORMで、エラーや例外が発生した場合にトランザクションがロールバックされる機能が提供されています。ただし、SQLAlchemyでは明示的に `session.rollback()` を呼び出す必要があります。

これらの違いを理解し、各ORMのドキュメントやベストプラクティスに従って適切にトランザクションを管理することが重要です。

# 比較が簡単にできて素晴らしい

全く知らない人には誤解を招きそうな表現(Active Recordで`ActiveRecord::Base.transaction`を明示的呼びたい場合は複数の操作をまとめたい時で、通常は`save`などで自動的にコミットされる)だが、少しでも知っている人はわかる書き方がされている。

# まとめ

- ChatGPTは、設計や問題解決のためのアプローチを多角的に考える手助けとして非常に有用。
- 技術の関連性を整理し、深く理解する作業がChatGPTを用いることで劇的に容易になる。