Guide Programmeur – TFDefect GitHub Action

Introduction

Ce guide s'adresse aux développeurs impliqués dans l'amélioration, l'extension ou la maintenance de l'application *TFDefect GitHub Action*. Il vise à fournir une documentation complète et approfondie sur la structure, le fonctionnement et les meilleures pratiques associées au développement de cette action GitHub. Ce document offre une vue détaillée de l'organisation du projet, présente le workflow CI/CD utilisé, décrit précisément les interactions entre les composants principaux, et énumère les bonnes pratiques techniques à suivre pour garantir un code clair, maintenable, testable et évolutif sur le long terme.

> Pour les instructions d'installation, d'utilisation ou de configuration, consultez le README.md.

Organisation du Projet

Structure des Répertoires

La structure suivante facilite la navigation dans le projet et reflète les principes fondamentaux de la *Clean Architecture*, en assurant une séparation claire des responsabilités logiques :

Racine

- `.github/`
  - `workflows/` : Contient la définition du pipeline CI/CD via GitHub Actions (notamment `ci.yml`).
  - `ISSUE_TEMPLATE/` : Fournit des modèles standardisés pour la création d'issues (bugs, documentation, fonctionnalités).
  - `PULL_REQUEST_TEMPLATE.md` : Sert de guide lors de la soumission d'une pull request.
- `app/`
  - `action_runner.py` : Point d'entrée principal, orchestre l'ensemble du processus d'analyse.
  - `config.py` : Centralise tous les paramètres et chemins configurables.
- `core/`
  - `use_cases/` : Contient les cas d'usage métiers spécifiques à l'action.
    - `detect_tf_changes.py`, `analyze_tf_code.py`, `feature_vector_builder.py`, `report_generator.py`
  - `parsers/` : Logique d'extraction et de traitement des métriques Terraform.
    - Inclut : `terraform_parser.py`, `metrics_extractor_factory.py`, `base_metrics_extractor.py`, ainsi que les extracteurs spécifiques.
- `infrastructure/`
  - `git/` : Gère les interactions avec le dépôt Git via des outils comme PyDriller.
    - Exemples : `git_adapter.py`, `git_changes.py`
  - `ml/` : Contient les modèles de prédiction ML utilisés pour détecter les bogues.

        - `base_model.py`, `dummy_model.py`, `model_factory.py`, `defect_history_manager.py`
- `templates/` : Fichiers HTML pour générer des rapports lisibles et interactifs.
- `libs/` : Librairies externes, par exemple `terraform_metrics-1.0.jar` (outil d'analyse Java).
- `tests/`
    - `unit/` : Tests unitaires.
    - `integration/` : Tests d'intégration complets.
- `data/` : Contient des fichiers Terraform utilisés pour la validation et les tests.
- `doc/` : Diagrammes d'architecture et documentation technique.
- `out/` : Dossier de sortie pour les résultats générés : rapports HTML, fichiers JSON, etc.
- `utils/` : Fonctions utilitaires partagées, réutilisables dans différents modules.

📄 Fichiers critiques

Fichier	Rôle
`action_runner.py`	Exécution principale de l'action GitHub
`detect_tf_changes.py`	Détection de blocs Terraform modifiés dans un commit
`feature_vector_builder.py`	Construction des vecteurs utilisés par le modèle ML
`dummy_model.py`	Modèle simple pour tests et démos
`report_generator.py`	Génère un rapport complet en HTML
`ci.yml`	Configuration du pipeline GitHub Actions

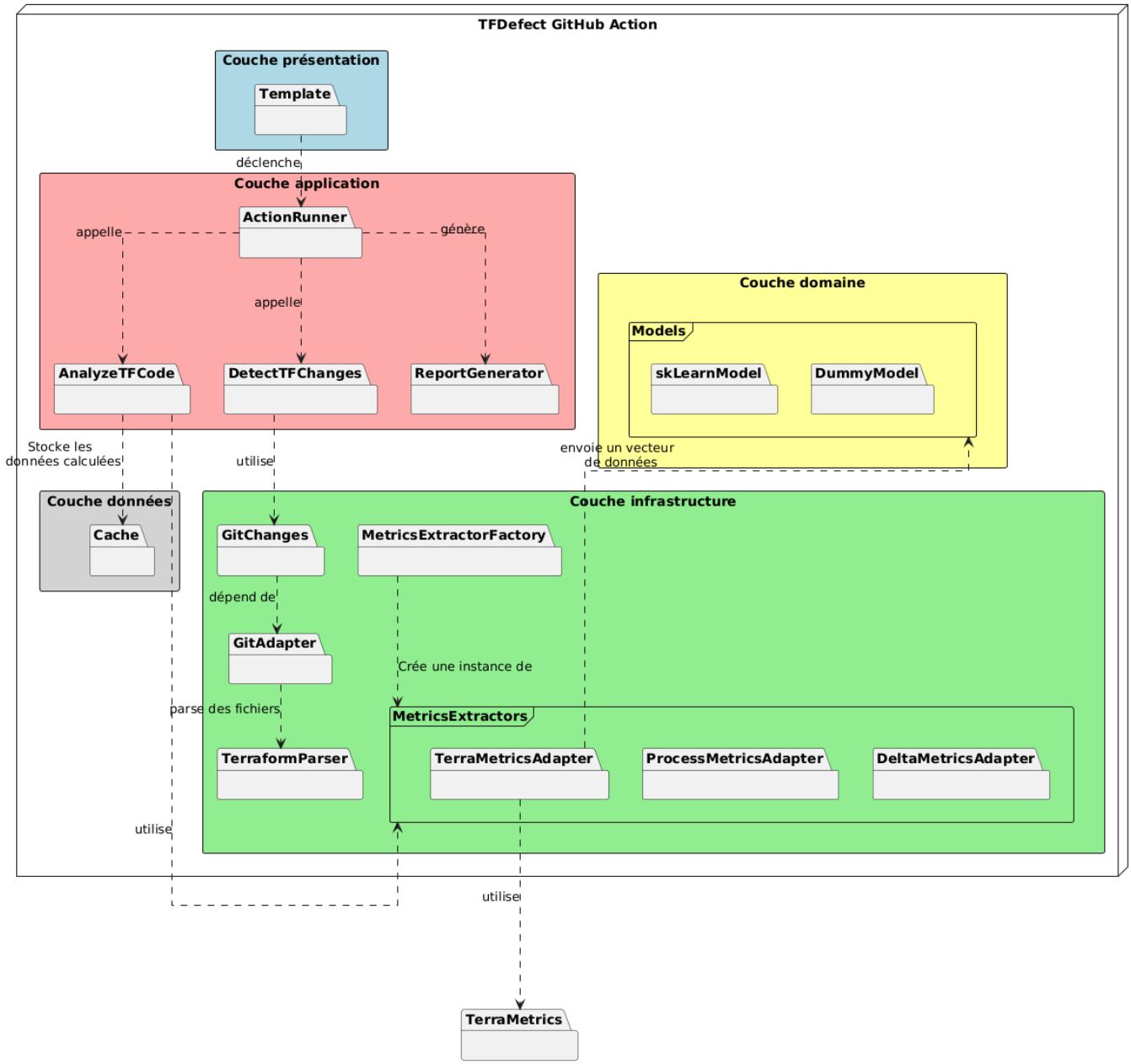
⚙️ Workflow CI/CD avec GitHub Actions

Le pipeline CI/CD décrit dans `.github/workflows/ci.yml` automatise les étapes suivantes à chaque push ou pull request sur la branche principale ou de fonctionnalité :

1. **Clonage du dépôt** avec ses sous-modules si présent
2. **Installation de Python**, configuration de l'environnement virtuel et des dépendances
3. **Installation conditionnelle de Java** (nécessaire pour TerraMetrics)
4. **Exécution automatique de TFDefect** avec les paramètres par défaut (modèle `dummy`, chemin des fichiers définis dans `config.py`)
5. **Vérification et sauvegarde des artefacts** (rapports HTML, historique des défauts JSON)
6. **Possibilité d'envoi de notifications** ou publication d'un commentaire sur la pull request

Ce workflow est conçu pour être facilement personnalisable, modulaire et exécutable localement si besoin (en simulant les étapes via un script).

🚀 Interaction des Composants



La logique d'exécution est basée sur l'orchestration par `action_runner.py`, qui appelle successivement les cas d'usage suivants :

1. `detect_tf_changes.py` utilise `git_changes.py` pour identifier les fichiers `.tf` modifiés dans le dernier commit.
2. `analyze_tf_code.py` décompose les blocs Terraform modifiés et passe les données à un ou plusieurs extracteurs via `metrics_extractor_factory.py`.
3. Chaque extracteur (ex. `code_metrics_extractor.py`, `delta_metrics_extractor.py`) retourne un ensemble de métriques.
4. Ces métriques sont structurées sous forme de vecteurs par `feature_vector_builder.py`.
5. Le vecteur est ensuite traité par le modèle prédictif choisi via `model_factory.py`.
6. `defect_history_manager.py` enregistre les résultats dans `defect_history.json`, enrichissant l'historique.
7. `report_generator.py` génère une synthèse lisible avec les blocs à risque, visualisable dans un navigateur.

Cette séparation des responsabilités rend le code testable, extensible et facilement modifiable.

🔧 Bonnes Pratiques Techniques

Afin d'assurer la stabilité, la qualité et la maintenabilité du projet, les pratiques suivantes sont fortement recommandées :

- ✅ Suivi des principes de la *Clean Architecture*
- ✅ Typage explicite grâce à `typing`, facilitant l'autocomplétion et la vérification statique
- ✅ Utilisation de `logging` pour journaliser les événements et faciliter le débogage
- ✅ Tests systématiques avec `pytest`, couverture de code recommandée > 80%
- ✅ Respect du style PEP8, mise en forme automatique avec `black`
- ✅ Modularité des composants : chaque module remplit une fonction claire
- ✅ Utilisation de `model_factory.py` pour garantir l'interchangeabilité des modèles ML
- ✅ Centralisation des chemins, paramètres, et options via `config.py`

🔢 Gestion des Configurations

Tous les paramètres critiques sont regroupés dans `app/config.py`. Cela inclut :

- Le chemin vers le fichier `.jar` de TerraMetrics
- Les emplacements de sortie des rapports et fichiers JSON
- Le modèle à utiliser par défaut (ex : `dummy`)
- Le chemin du dépôt local analysé
- Les options activables par développeur pour ajuster le comportement de l'action

Cette centralisation évite la duplication de logique de configuration dans plusieurs fichiers.

🎉 Conclusion

Cette documentation fournit une vision d'ensemble rigoureuse et concrète du projet TFDefectGA. Elle permet aux développeurs de comprendre en profondeur le fonctionnement interne de l'application, de participer efficacement à son évolution, et de contribuer à sa qualité logicielle.

En adoptant une architecture modulaire et testable, le projet favorise la contribution ouverte, l'évolutivité des fonctionnalités, et une gestion efficace de la dette technique.

> 📄 Pensez à maintenir ce guide à jour à chaque évolution significative du projet.