Module 3

Pipelines d'Intégration Continue (CI)

Principes de l'Intégration Continue

Intégration Fréquente

Intégrer le code plusieurs fois par jour pour détecter rapidement les conflits

Automatisation

Automatiser les builds, tests et déploiements pour réduire les erreurs humaines

Tests Automatisés

Exécuter automatiquement les tests à chaque intégration pour garantir la qualité

Feedback Rapide

Obtenir un retour immédiat sur l'état du build et des tests

Azure Pipelines : Vue d'ensemble

Service cloud pour créer et gérer des pipelines CI/CD automatisés, supportant multiple langages et plateformes

Cloud Native

Service entièrement géré dans Azure

Multi-langages

Support .NET, Java, Python, Node.js

Multi-plateformes

Windows, Linux, macOS

Composants d'un Pipeline CI

Agents

Machines virtuelles qui exécutent les tâches du pipeline

Tâches (Tasks)

Unités d'exécution individuelles (build, test, deploy)

Étapes (Steps)

Séquence de tâches exécutées dans un ordre défini

Jobs

Collection d'étapes exécutées sur le même agent

Phases (Stages)

Groupes logiques de jobs (Build, Test, Deploy)

Configuration d'un Pipeline CI (YAML)

azure-pipelines.yml

```yaml
trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: DotNetCoreCLI@2
  inputs:
    command: 'build'
    projects: '**/*.csproj'
```

trigger

Définit quand le pipeline s'exécute

pool

Spécifie l'agent d'exécution

steps

Liste des tâches à exécuter

task

Tâche prédéfinie avec paramètres

Déclencheurs de Pipeline

CI Trigger

Déclenché automatiquement lors des commits sur les branches spécifiées

PR Trigger

Déclenché lors de la création ou mise à jour d'une Pull Request

Scheduled

Déclenché selon un planning défini (cron expression)

Filtres Disponibles

Filtres de branches

Filtres de chemins

Filtres de tags

Intégration des Tests Automatisés

Importance en CI

- Détection précoce des régressions
- Garantie de qualité du code
- Feedback immédiat aux développeurs
- Confiance pour les déploiements

Tests Unitaires

Tests isolés des composants individuels du code

Tests d'Intégration

Tests des interactions entre composants

Tests Fonctionnels

Tests de bout en bout du comportement utilisateur

Outils de Qualité du Code

SonarQube / SonarCloud

Plateforme leader pour l'analyse continue de la qualité du code, détectant bugs, vulnérabilités et code smells

Bugs

Détection d'erreurs dans le code

Vulnérabilités

Identification des failles de sécurité

Code Smells

Problèmes de maintenabilité

Couverture

Pourcentage de code testé

Gestion des Dépendances

Restauration de Packages

Téléchargement automatique des dépendances nécessaires au build

Gestionnaires de Packages

NuGet (.NET), npm (Node.js), Maven (Java), pip (Python)

Azure Artifacts

Flux de packages privés pour héberger et partager des packages

Gestion des Versions

Contrôle des versions de dépendances pour la reproductibilité

Gestion des Artefacts

Build

Compilation du code source

Publish

Publication des artefacts

Consume

Consommation par d'autres pipelines

Build Artifacts

Fichiers compilés, packages, binaires produits par le build

Test Results

Rapports de tests, couverture de code, métriques qualité

Variables et Groupes de Variables

$ Types de Variables

Variables Prédéfinies

$(Build.SourceBranch), $(Agent.OS)

Variables Personnalisées

Définies dans le pipeline YAML

Variables d'Environnement

Variables système de l'agent

```
variables:
  buildConfiguration: 'Release'
  solution: '**/*.sln'
```

Groupes de Variables

Collections de variables réutilisables entre pipelines, avec gestion des secrets et permissions

Variables Secrètes

Variables sensibles chiffrées, non visibles dans les logs

Variables Runtime

Variables calculées ou modifiées pendant l'exécution

Conditions et Boucles dans les Pipelines

Conditions

Condition sur Branche

```
condition: eq(variables['Build.SourceBranch'],
'refs/heads/main')
```

Condition sur Succès

```
condition: succeeded()
```

Condition Personnalisée

```
condition: and(succeeded(),
eq(variables['deployToProduction'], 'true'))
```

Stratégie Each

Exécution parallèle pour chaque élément d'une liste

```
strategy:
matrix:
Python37:
python.version: '3.7'
Python38:
python.version: '3.8'
```

Retry et Timeout

Contrôle des tentatives et délais d'expiration

```
- task: MyTask@1
retryCountOnTaskFailure: 3
timeoutInMinutes: 10
```

Meilleures Pratiques CI

Builds Rapides

Optimiser les temps de build pour un feedback rapide (< 10 minutes)

Intégration Fréquente

Intégrer le code plusieurs fois par jour sur la branche principale

Tests Automatisés

Couverture de tests élevée avec tests unitaires et d'intégration

Fail Fast

Arrêter le pipeline dès la première erreur pour économiser les ressources

Builds Reproductibles

Utiliser des versions fixes des dépendances et outils

Monitoring

Surveiller les métriques de performance et de qualité

Dépannage des Pipelines CI

Analyse des Logs

Examiner les logs détaillés pour identifier les erreurs et points de blocage

Débogage Local

Reproduire les problèmes en local avec les mêmes conditions que le pipeline

Problèmes d'Agent

Vérifier la disponibilité et configuration des agents d'exécution

Connectivité

Diagnostiquer les problèmes de réseau et d'accès aux ressources externes

Timeouts

Ajuster les délais d'expiration pour les tâches longues

Permissions

Vérifier les droits d'accès aux dépôts, secrets et ressources

Récapitulatif du Module 3

Concepts CI

Principes de l'intégration continue et composants des pipelines

Azure Pipelines

Configuration YAML, agents, tâches et déclencheurs

Tests & Qualité

Intégration des tests automatisés et outils de qualité du code

Artefacts

Gestion des dépendances et publication d'artefacts

Contrôle de Flux

Variables, conditions et boucles dans les pipelines

Bonnes Pratiques

Optimisation, dépannage et maintenance des pipelines CI