Une image contenant texte, Police, logo, conception

Description générée automatiquement

**PROJET 2023-2024**

|  |
| --- |
| **Conception d’une application de météo** |

**Ingénierie Informatique et Réseaux**

Réalisé par :

ELKASTALI Otmane

AMIMI Yahya

Mohamed Karim Touzani

Introduction Générale

La météo joue un rôle prépondérant dans notre quotidien, influençant nos choix et nos activités. Afin de répondre aux besoins croissants d'informations météorologiques précises et accessibles, nous avons entrepris le développement d'une application dédiée à fournir une expérience utilisateur immersive et fiable. Cette application météo vise à simplifier l'accès aux données météorologiques les plus récentes, tout en offrant des fonctionnalités avancées pour aider les utilisateurs à mieux planifier leurs journées.

Au cours de ce projet, nous avons cherché à combiner la technologie moderne avec la précision des prévisions météorologiques, afin de créer une application qui va au-delà des simples prévisions basiques. Notre objectif était de fournir une interface intuitive et conviviale, permettant aux utilisateurs de visualiser rapidement les conditions météorologiques actuelles, ainsi que de planifier à l'avance en fonction des prévisions à court et à long terme.

Ce rapport détaillera le processus de conception, de développement et de mise en œuvre de l'application météo, en mettant en lumière les choix technologiques, les défis rencontrés et les fonctionnalités clés intégrées. De plus, il examinera l'expérience utilisateur, les retours des utilisateurs, et les pistes d'amélioration futures pour continuer à évoluer dans le paysage dynamique des applications météorologiques.

## Introduction

**Aperçu du Projet :**

Notre projet consiste en le développement d'une application météo innovante visant à fournir des informations météorologiques précises et accessibles. Nous avons mis l'accent sur la combinaison de la technologie moderne avec la précision des prévisions météorologiques, tout en offrant une expérience utilisateur immersive. L'application se distingue par son interface intuitive, ses fonctionnalités avancées, et sa capacité à répondre aux besoins des utilisateurs en matière de visualisation rapide des conditions actuelles et de planification à court et long terme.

**Importance de l'Architecture Microservices :**

L'architecture microservices joue un rôle essentiel dans le développement et le déploiement réussi de notre application météo. Voici quelques points clés illustrant l'importance de cette approche :

**Évolutivité** : L'architecture microservices permet une évolutivité plus facile en permettant le déploiement, la mise à jour et la maintenance indépendants de chaque composant. Cela facilite l'ajout de nouvelles fonctionnalités sans perturber l'ensemble de l'application.

**Flexibilité** : Chaque microservice est une entité autonome, ce qui offre une flexibilité considérable pour le choix des technologies spécifiques à chaque composant. Cela favorise l'utilisation des meilleures technologies adaptées à chaque service, améliorant ainsi la performance globale de l'application.

**Réactivité** : Les microservices permettent une réponse rapide aux changements et aux mises à jour. Cela est particulièrement crucial dans le contexte des prévisions météorologiques, où la rapidité de l'actualisation des données est cruciale pour fournir des informations précises aux utilisateurs.

**Gestion Simplifiée** : La gestion indépendante des microservices facilite la détection et la résolution des problèmes. Les équipes de développement peuvent se concentrer sur des services spécifiques sans être contraintes par la complexité globale de l'application.

**Scalabilité** : L'architecture microservices permet une scalabilité horizontale, c'est-à-dire l'ajout de nouvelles instances de microservices pour gérer une charge croissante. Cela garantit une performance constante même en cas d'augmentation significative du nombre d'utilisateurs.

**Conclusion**

L’adoption de l'architecture microservices a été cruciale pour le succès de notre projet météo. Elle nous a permis de créer une application robuste, évolutive et réactive, répondant ainsi aux attentes élevées des utilisateurs en matière de précision et de convivialité.

**2. Architecture Microservices**

* **Architecture**

L'illustration ci-jointe offre une vue d'ensemble détaillée de l'architecture de notre application, mettant en lumière la structure robuste et les relations entre les différents composants pour assurer une mise en œuvre efficace de nos fonctionnalités.

Une image contenant texte, capture d’écran, diagramme, cercle

Description générée automatiquement

Figure 1 architecture de l’application

* **Description des services**

Une image contenant capture d’écran, texte, diagramme, ligne

Description générée automatiquement

Figure 2 Communication entre le microservices

**Microservice Geocoder :**

**Fonction Principale** : Le microservice Geocoder est responsable de la conversion des adresses (par exemple, "Marrakech") en coordonnées géographiques (latitude et longitude).

**Entrées** : Adresse en format texte.

**Sortie** : Coordonnées géographiques (latitude et longitude).

**Fonctionnement** : Lorsqu'il reçoit une adresse en entrée, le Geocoder effectue une recherche géographique et renvoie les coordonnées correspondantes.

**Microservice Weather Service :**

**Fonction Principale** : Ce microservice fournit les conditions météorologiques actuelles pour une adresse donnée.

**Entrées** : Adresse en format texte.

**Sortie** : Informations météorologiques en temps réel (température, humidité, vitesse du vent, etc.).

**Fonctionnement** : Le Weather Service reçoit l'adresse, l'envoie au microservice Geocoder pour obtenir les coordonnées, puis interroge une source météorologique pour récupérer les données actuelles.

**Microservice WeatherForecast Service :**

**Fonction Principale** : Ce microservice fournit des prévisions météorologiques détaillées sur une période de 5 jours, avec des mises à jour toutes les 3 heures.

**Entrées** : Adresse en format texte.

**Sortie** : Prévisions météorologiques pour les 5 prochains jours, avec des intervalles de 3 heures.

**Fonctionnement** : Le WeatherForecast Service utilise l'adresse fournie, la soumet au microservice Geocoder pour obtenir les coordonnées, puis interroge une source météorologique pour récupérer les prévisions détaillées.

**Interactions entre les Microservices :**

L'utilisateur soumet une adresse au microservice Weather Service.

Le Weather Service envoie cette adresse au microservice Geocoder pour obtenir les coordonnées géographiques correspondantes.

Une fois les coordonnées obtenues, le Weather Service interroge la source météorologique pour récupérer les conditions météorologiques actuelles et les renvoie à l'utilisateur.

En parallèle, l'adresse est également envoyée au WeatherForecast Service, qui suit le même processus pour obtenir les prévisions météorologiques à long terme.

L'utilisateur reçoit ensuite à la fois les informations météorologiques actuelles et les prévisions, toutes obtenues à partir de l'adresse initiale soumise.

**3. Conception des Microservices**

La conception des microservices pour notre projet météo repose sur une approche modulaire visant à offrir une flexibilité, une évolutivité et une maintenance optimales. Voici les principaux aspects de la conception des microservices pour notre application :

**Décomposition Fonctionnelle** : Nous avons identifié les principales fonctionnalités de notre application météo, telles que la géocodification, la fourniture de données météorologiques actuelles, et la prévision météorologique à long terme. Chaque fonctionnalité est attribuée à un microservice spécifique.

**Délimitation des Responsabilités** : Chaque microservice a une responsabilité claire et autonome. Le microservice Geocoder est dédié à la conversion d'adresses en coordonnées géographiques, le Weather Service fournit les conditions météorologiques actuelles, et le WeatherForecast Service donne des prévisions à long terme.

**Interfaces et Contrats** : Les interfaces entre les microservices sont clairement définies. Des contrats (API) précis régissent la manière dont les microservices interagissent. Par exemple, le Weather Service envoie une adresse au Geocoder et utilise ensuite les coordonnées obtenues pour récupérer les données météorologiques.

**Choix des Technologies** : Chaque microservice utilise des technologies adaptées à ses besoins spécifiques. Le Geocoder peut utiliser des services de géocodification existants, le Weather Service interagit avec des sources météorologiques, et le WeatherForecast Service utilise des algorithmes de prévision.

**Communication entre Microservices** : La communication entre les microservices se fait principalement par le biais d'appels d'API bien définis. Les mécanismes peuvent inclure l'utilisation de protocoles HTTP/REST pour assurer une communication efficace.

**Gestion des Données** : La gestion des données est soigneusement planifiée. Chaque microservice peut stocker des données spécifiques nécessaires à sa responsabilité, tout en partageant les informations essentielles avec les autres microservices via des mécanismes appropriés.

**Évolutivité et Résilience** : Les mécanismes d'évolutivité et de résilience sont intégrés à la conception. Chaque microservice peut être déployé et mis à jour indépendamment, garantissant une évolutivité aisée et une résilience face à des pannes partielles.

Cette approche de conception des microservices vise à créer une application météo flexible, extensible et facile à maintenir, répondant ainsi aux exigences d'une expérience utilisateur optimale dans le domaine de l'information météorologique

**4.Conteneurisation avec Docker**

**DockerFile pour les applications spring boot :**

**FROM** openjdk:17-oracle  
**VOLUME /**tmp  
**COPY** target**/\***.jar app.jar  
**ENTRYPOINT** ["java","-jar","/app.jar"]

Ce Dockerfile prend une image de base OpenJDK 17 d'Oracle, crée un point de montage dans le conteneur, copie le fichier JAR généré dans le répertoire cible du projet, puis configure l'entrée principale pour démarrer l'application Java à partir de ce fichier JAR lors du lancement du conteneur. Ce type de configuration est couramment utilisé pour créer des conteneurs Docker prêts à exécuter des applications Java autonomes.

**DockerFile pour les applications React :**

**FROM** node:latest  
**WORKDIR /**app  
**COPY** package.json .  
**RUN** npm install  
**COPY** . .  
**CMD** ["npm", "start"]

Ce Dockerfile prend l'image officielle Node.js comme base, configure un répertoire de travail, copie le fichier "package.json" pour installer les dépendances, copie tous les fichiers du projet, puis définit la commande par défaut pour lancer l'application Node.js lors du démarrage du conteneur. Ce type de configuration est fréquemment utilisé pour créer des images Docker pour des applications Node.js.

**Docker-Compose :**

**discovery-service**:  
 **container\_name**: discovery-service  
 **build**: ./discovery-service  
 **ports**:  
 - "8761:8761"  
 **expose**:  
 - "8761"  
 **healthcheck**:  
 **test**: ["CMD", "curl", "-f", "http://localhost:8761/actuator/health"]  
 **interval**: 10s  
 **retries**: 3

**Service discovery-service :**

**Objectif** : Ce service représente le service de découverte (Discovery Service) basé sur Eureka.

**Configuration** : Il est construit à partir du répertoire ./discovery-service, expose le port 8761 pour les communications, et vérifie la santé du service à l'aide d'une requête HTTP vers l'endpoint /actuator/health.

**Dépendances** : Aucune dépendance explicite, car le Discovery Service agit généralement comme un point central de découverte pour les autres services.

**config-service**:  
 **build**: ./config-service  
 **container\_name**: config-service  
 **ports**:  
 - '9999:9999'  
 **expose**:  
 - '9999'  
 **environment**:  
 **DISCOVERY\_SERVICE\_URL**: http://discovery-service:8761/eureka/  
 **healthcheck**:  
 **test**: [ "CMD", "curl", "-f", "http://localhost:9999/actuator/health" ]  
 **interval**: 10s  
 **retries**: 4  
 **depends\_on**:  
 **discovery-service**:  
 **condition**: service\_healthy

**Service config-service :**

**Objectif** : Il s'agit du service de configuration qui gère la configuration centralisée des autres services.

**Configuration** : Construit à partir du répertoire ./config-service, expose le port 9999, et vérifie la santé du service via une requête HTTP vers l'endpoint /actuator/health.

**Dépendances** : Dépend du discovery-service pour la découverte des services et s'assure que le service est sain avant de démarrer.

**geocoder-service**:  
 **container\_name**: geocoder-service  
 **build**: ./geocoder-service  
 **ports**:  
 - "8082:8082"  
 **expose**:  
 - "8082"  
 **depends\_on**:  
 **discovery-service**:  
 **condition**: service\_healthy  
 **config-service**:  
 **condition**: service\_healthy  
 **environment**:  
 **DISCOVERY\_SERVICE\_URL**: http://discovery-service:8761/eureka/  
 **CONFIG\_SERVICE**: http://config-service:9999/  
 **healthcheck**:  
 **test**: [ "CMD", "curl", "-f", "http://localhost:8082/actuator/health" ]  
 **interval**: 10s  
 **retries**: 3

**Service geocoder-service :**

**Objectif** : Représente le service de géocodage qui convertit des adresses en coordonnées géographiques.

**Configuration** : Construit à partir du répertoire ./geocoder-service, expose le port 8082, et vérifie la santé à travers l'endpoint /actuator/health.

**Dépendances** : Dépend du discovery-service et du config-service pour la découverte des services et s'assure que ces services sont sains avant de démarrer.

**weather-service**:  
 **container\_name**: weather-service  
 **build**: ./weather-service  
 **ports**:  
 - "8081:8081"  
 **expose**:  
 - "8081"  
 **depends\_on**:  
 **discovery-service**:  
 **condition**: service\_healthy  
 **config-service**:  
 **condition**: service\_healthy  
 **environment**:  
 **DISCOVERY\_SERVICE\_URL**: http://discovery-service:8761/eureka/  
 **CONFIG\_SERVICE**: http://config-service:9999/  
 **healthcheck**:  
 **test**: [ "CMD", "curl", "-f", "http://localhost:8081/actuator/health" ]  
 **interval**: 10s  
 **retries**: 3

**Service weather-service :**

**Objectif** : Service fournissant les conditions météorologiques actuelles.

**Configuration** : Construit à partir du répertoire ./weather-service, expose le port 8081, et vérifie la santé à travers l'endpoint /actuator/health.

**Dépendances** : Dépend du discovery-service et du config-service pour la découverte des services et s'assure que ces services sont sains avant de démarrer.

**weather-forecast**:  
 **container\_name**: weather-forecast  
 **build**: ./weather-forecast  
 **ports**:  
 - "8083:8083"  
 **expose**:  
 - "8083"  
 **depends\_on**:  
 **discovery-service**:  
 **condition**: service\_healthy  
 **config-service**:  
 **condition**: service\_healthy  
 **environment**:  
 **DISCOVERY\_SERVICE\_URL**: http://discovery-service:8761/eureka/  
 **CONFIG\_SERVICE**: http://config-service:9999/  
 **healthcheck**:  
 **test**: [ "CMD", "curl", "-f", "http://localhost:8083/actuator/health" ]  
 **interval**: 10s  
 **retries**: 3

**Service weather-forecast :**

**Objectif** : Service fournissant les prévisions météorologiques pour les 5 prochains jours.

**Configuration** : Construit à partir du répertoire ./weather-forecast, expose le port 8083, et vérifie la santé à travers l'endpoint /actuator/health.

**Dépendances** : Dépend du discovery-service et du config-service pour la découverte des services et s'assure que ces services sont sains avant de démarrer.

**gateway-service**:  
 **container\_name**: gateway-service  
 **build**: ./gateway-service  
 **ports**:  
 - "8888:8888"  
 **expose**:  
 - "8888"  
 **depends\_on**:  
 **discovery-service**:  
 **condition**: service\_healthy  
 **config-service**:  
 **condition**: service\_healthy  
 **environment**:  
 **DISCOVERY\_SERVICE\_URL**: http://discovery-service:8761/eureka/  
 **CONFIG\_SERVICE**: http://config-service:9999/

**Service gateway-service :**

**Objectif** : Service de passerelle (API Gateway) qui agit comme point d'entrée principal pour l'application.

**Configuration** : Construit à partir du répertoire ./gateway-service, expose le port 8888, et vérifie la santé à travers l'endpoint /actuator/health.

**Dépendances** : Dépend du discovery-service et du config-service pour la découverte des services et s'assure que ces services sont sains avant de démarrer.

**weather-ui**:  
 **container\_name**: weather-ui  
 **build**: ./weather\_app\_front  
 **ports**:  
 - "3000:3000"  
 **expose**:  
 - "3000"  
 **stdin\_open**: true  
 **tty**: true  
 **depends\_on**:  
 **gateway-service**:  
 **condition**: service\_started

**Service weather-ui :**

**Objectif** : Service front-end représentant l'interface utilisateur de l'application météo.

**Configuration** : Construit à partir du répertoire ./weather\_app\_front, expose le port 3000, et déclare une dépendance envers le gateway-service, s'assurant que ce service a démarré avant de commencer.

**Dépendances** : Dépend du gateway-service pour l'accès aux fonctionnalités météorologiques via l'API Gateway.

**5.CI/CD avec Jenkins**

**Intégration Continue (CI) :**

* À chaque modification du code source dans votre référentiel GitHub, Jenkins est automatiquement notifié via le webhook.
* Jenkins récupère le code source mis à jour depuis le référentiel.
* Les dépendances sont installées, et le code est compilé pour garantir son intégrité.
* Des tests unitaires et d'intégration sont exégrés pour valider la stabilité du code.

**Déploiement Continu (CD) :**

* Après un succès du processus CI, une étape d'analyse du code est intégrée au pipeline de déploiement continu.
* SonarQube est utilisé pour effectuer une analyse statique du code source, identifiant ainsi les violations potentielles des normes de codage, les bugs, les vulnérabilités de sécurité, et les codes dupliqués.
* SonarQube fournit des rapports détaillés sur la qualité du code, offrant une vision claire des problèmes identifiés et des zones à améliorer.
* Les résultats de l'analyse SonarQube sont intégrés au tableau de bord du pipeline Jenkins, offrant une visibilité immédiate sur la qualité du code et les éventuelles violations des normes de codage.
* En cas de succès des tests, le pipeline de déploiement continu déclenche automatiquement le déploiement de l'application météo sur Docker.
* SonarQube génère des rapports post-déploiement qui permettent d'évaluer l'impact des nouvelles versions sur la qualité globale du code et d'identifier les domaines nécessitant une attention particulière.

stages {  
 stage('Git Clone') {  
 steps {  
 script {  
 checkout([$class: 'GitSCM', branches: [[name: 'main']], userRemoteConfigs: [[url: 'https://github.com/DARR3NG/weatherApp-docker-jenkins.git']]])  
  
 }  
 }  
 }

**Stage 'Git Clone' :**

* Cette étape récupère le code source du référentiel GitHub (https://github.com/DARR3NG/weatherApp-docker-jenkins.git) en utilisant Git et le clone dans l'espace de travail de Jenkins.

stage('Build Discovery-Service') {  
 steps {  
 script{  
 dir('weather-app/discovery-service') {  
  
 withSonarQubeEnv('sonarserver') {  
 bat "mvn clean verify sonar:sonar -Dsonar.projectKey=discovery-service -Dsonar.projectName='discovery-service'"  
 }  
 timeout(time:1, unit:'HOURS'){  
 **def** qg **=** waitForQualityGate()  
 **if** (qg**.**status **!=** 'OK') {  
 error "Quality Gate did not pass. Check the SonarQube dashboard for details."  
 }  
 }  
 bat 'mvn clean install'  
 }  
 }  
 }  
}

**Stage 'Build Discovery-Service' :**

* Le code du service de découverte (discovery-service) est récupéré depuis le référentiel.
* une installation propre du projet est réalisée

stage('Build Config-Service') {  
 steps {  
 dir('weather-app/config-service'){  
  
 bat 'mvn clean install'  
 }  
 }  
}

**Stage 'Build Config-Service' :**

* Le code du service de configuration (config-service) est récupéré depuis le référentiel.
* Le service est ensuite compilé et testé.
* Enfin, une installation propre du projet est réalisée.

**Stage 'Build gateway-service' :**

stage('Build gateway-service') {  
 steps {  
 dir('weather-app/gateway-service'){  
  
 bat 'mvn clean install'  
 }  
 }  
}

* Le code du service de passerelle (gateway-service) est récupéré depuis le référentiel.
* Le service est ensuite compilé et testé.
* Enfin, une installation propre du projet est réalisée.

stage('Build geocoder-service') {  
 steps {  
 script{  
 dir('weather-app/geocoder-service'){  
  
 withSonarQubeEnv('sonarserver') {  
 bat "mvn clean verify sonar:sonar -Dsonar.projectKey=geocoder-service -Dsonar.projectName='geocoder-service'"  
 }  
  
 timeout(time:1, unit:'HOURS'){  
 **def** qg **=** waitForQualityGate()  
 **if** (qg**.**status **!=** 'OK') {  
 error "Quality Gate did not pass. Check the SonarQube dashboard for details."  
 }  
 }  
  
 bat 'mvn clean install'  
 }  
 }  
 }  
}

**Stage 'Build geocoder-service' :**

* Le code du service de géocodage (geocoder-service) est récupéré depuis le référentiel.
* Une analyse SonarQube est effectuée pour évaluer la qualité du code.
* Si la qualité est satisfaisante, le service est ensuite compilé et testé.
* Enfin, une installation propre du projet est réalisée.

stage('Build weather-service') {  
 steps {  
 script{  
 dir('weather-app/weather-service'){  
  
 withSonarQubeEnv('sonarserver') {  
 bat "mvn clean verify sonar:sonar -Dsonar.projectKey=weather-service - Dsonar.projectName='weather-service'"  
 }  
 timeout(time:1, unit:'HOURS'){  
 **def** qg **=** waitForQualityGate()  
 **if** (qg**.**status **!=** 'OK') {  
 error "Quality Gate did not pass. Check the SonarQube dashboard for details."  
 }  
 }  
  
 bat 'mvn clean install'  
 }}  
 }  
}

**Stage 'Build weather-service' :**

* Le code du service météorologique (weather-service) est récupéré depuis le référentiel.
* Une analyse SonarQube est effectuée pour évaluer la qualité du code.
* Si la qualité est satisfaisante, le service est ensuite compilé et testé.
* Enfin, une installation propre du projet est réalisée.

stage('Build weather-forecast-service') {  
 steps {  
 script{  
 dir('weather-app/weather-forecast'){  
  
 withSonarQubeEnv('sonarserver') {  
 bat "mvn clean verify sonar:sonar -Dsonar.projectKey=weather-forecast -Dsonar.projectName='weather-forecast'"  
 }  
 timeout(time:1, unit:'HOURS'){  
 **def** qg **=** waitForQualityGate()  
 **if** (qg**.**status **!=** 'OK') {  
 error "Quality Gate did not pass. Check the SonarQube dashboard for details."  
 }  
 }  
  
 bat 'mvn clean install'  
 }  
 }  
 }  
}

**Stage 'Build weather-forecast-service' :**

* Le code du service de prévisions météorologiques (weather-forecast) est récupéré depuis le référentiel.
* Une analyse SonarQube est effectuée pour évaluer la qualité du code.
* Si la qualité est satisfaisante, le service est ensuite compilé et testé.
* Enfin, une installation propre du projet est réalisée.

stage('Build Weather ui') {  
 steps {  
 dir('weather-app/weather\_app\_front'){  
 bat 'npm install'  
 }  
 }  
}

**Stage 'Build Weather ui' :**

* Le code de l'interface utilisateur météo (weather\_app\_front) est récupéré depuis le référentiel.
* Les dépendances nécessaires sont installées en utilisant npm.

stage('Compose and Build Docker Images') {  
 steps {  
 dir('weather-app'){  
  
 bat 'docker compose up --build -d'  
 }  
 }  
}

**Stage 'Compose and Build Docker Images' :**

* Le code de l'ensemble des services est récupéré.
* Un script Docker compose est utilisé pour construire et lancer les conteneurs Docker correspondant à chaque service.

Chaque étape intègre des phases d'analyse statique du code avec SonarQube et des tests pour assurer la qualité du code et minimiser les erreurs. Le pipeline prend en charge la construction, les tests, la qualité du code, et le déploiement des services et de l'interface utilisateur, offrant ainsi une approche complète du développement, de la validation et du déploiement continu pour votre application météo.

**6. Déploiement Automatique**

**Utilisation de Ngrok :**

**Exposition Sécurisée** : Ngrok permet d'exposer nos services locaux de manière sécurisée, en créant des tunnels chiffrés qui garantissent la confidentialité des données échangées.

**Accessibilité Externe** : Grâce à Ngrok, vos services locaux deviennent accessibles depuis Internet, ce qui peut être utile pour partager des fonctionnalités en cours de développement avec des collègues ou des clients, ou même pour tester des intégrations avec des services externes.

**Intégration avec CI/CD** : Ngrok peut être intégré dans les pipelines de déploiement continu (CI/CD), ce qui permet d'automatiser le processus d'exposition des services lors des phases de développement et de tests.

**7. Intégration de SonarQube**

L'intégration de SonarQube dans le projet météo offre un moyen puissant d'analyser et d'améliorer la qualité du code. Voici comment cette intégration a été configurée :

**Installation et Configuration** : SonarQube a été installé sur un serveur dédié, et la configuration a été adaptée aux besoins spécifiques du projet météo, notamment en définissant les règles de qualité et les seuils de conformité.

**Intégration avec le Processus de Build** : SonarQube a été intégré au processus de build automatisé du projet. Après chaque build, les résultats de l'analyse statique du code sont envoyés à SonarQube pour évaluation.

**Configuration :**

Création du webHooks dans sonarQube :

**Une image contenant capture d’écran, texte

Description générée automatiquement**

Configuration avec Jenkins, il faut d’abord installe le plugin SonarQube Scanner :

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Une image contenant texte, ligne, capture d’écran

Description générée automatiquement

**Modification du pipeline :**

stage('Build geocoder-service') {  
 steps {  
 script{  
 dir('weather-app/geocoder-service'){  
  
 withSonarQubeEnv('sonarserver') {  
 bat "mvn clean verify sonar:sonar -Dsonar.projectKey=geocoder-service -Dsonar.projectName='geocoder-service'"  
 }  
  
 timeout(time:1, unit:'HOURS'){  
 **def** qg **=** waitForQualityGate()  
 **if** (qg**.**status **!=** 'OK') {  
 error "Quality Gate did not pass. Check the SonarQube dashboard for details."  
 }  
 }  
  
 bat 'mvn clean install'  
 }  
 }  
 }  
}

Cette étape construit le service géocodeur du projet météo, l'analyse avec SonarQube pour évaluer la qualité du code, et attend que le Quality Gate soit satisfait. Si le Quality Gate est satisfait, la construction continue en installant les dépendances nécessaires. Sinon, une erreur est signalée.

**8. Conclusion**

**Résumé des Accomplissements :**

Le développement de l'application météo basée sur l'architecture microservices a été une entreprise réussie, marquée par plusieurs accomplissements significatifs :

**Conception et Développement** : Une architecture microservices a été adoptée, offrant une modularité, une évolutivité et une maintenance améliorées. Les microservices tels que le Geocoder, Weather Service et WeatherForecast Service ont été conçus pour fonctionner de manière autonome tout en collaborant harmonieusement.

**Intégration de SonarQube** : SonarQube a été intégré au processus de build, offrant une analyse statique du code et contribuant à l'amélioration continue de la qualité du code. Les règles personnalisées et la détection précoce des problèmes ont renforcé la robustesse du projet.

**Perspectives :**

**Intégration de Nouvelles Fonctionnalités** : L'exploration de nouvelles fonctionnalités, telles que des alertes météorologiques personnalisées, des intégrations avec des services tiers, ou des visualisations plus avancées, ouvrira de nouvelles opportunités pour l'application.

**Optimisation des Performances** : L'optimisation continue des performances, tant au niveau des microservices individuels que de l'application dans son ensemble, assurera une réactivité accrue et une expérience utilisateur sans heurts, même avec une augmentation du nombre d'utilisateurs.

**Élargissement de la Portée Géographique** : L'extension de la couverture géographique de l'application pour inclure davantage de régions et de localités élargira son utilité et sa pertinence pour un public plus large.