Docu$_t$*echnique*

AHMED HABBANI

December 2025

1 Phase 1.1 : Collecte des métriques système

Cette phase constitue la première étape de l'implémentation du système. Elle vise à mettre en place un mécanisme fiable de collecte des métriques système locales au niveau des agents de surveillance, indépendamment de toute communication réseau.

1.1 Objectif

L'objectif principal de cette phase est de disposer d'un collecteur de métriques capable de fournir, à un instant donné, un aperçu de l'état global de la machine sur laquelle l'agent est déployé. Les métriques collectées serviront de base aux phases ultérieures, notamment pour l'envoi des données au serveur central et la détection des alertes critiques.

1.2 Métriques collectées

Les métriques système suivantes ont été retenues :

- l'utilisation globale du processeur (CPU) ;
- l'utilisation de la mémoire physique (RAM) ;
- l'occupation de l'espace disque.

Chaque mesure est associée à un horodatage précis, permettant une exploitation ultérieure des données sous forme d'historique et de séries temporelles.

1.3 Implémentation

La collecte des métriques a été implémentée en **Java pur**, sans dépendance externe, en s'appuyant sur les API standard de la plateforme Java. L'accès aux informations système est réalisé à l'aide de l'interface `OperatingSystemMXBean`, fournie par le module de gestion JMX.

Les métriques sont encapsulées dans une classe dédiée représentant un échantillon de données, garantissant une structure claire, immuable et facilement exploitable lors des phases suivantes de l'implémentation.

Certaines méthodes utilisées étant marquées comme dépréciées depuis les versions récentes de Java, leur usage est explicitement documenté et encapsulé afin de préserver la maintenabilité du code tout en garantissant la compatibilité avec les environnements actuels.

1.4 Tests et validation

Afin de valider le bon fonctionnement du collecteur, un programme de test simple a été développé. Ce programme instancie le collecteur de métriques et affiche périodiquement les valeurs collectées dans la console.

Les tests réalisés ont permis de vérifier :

- la cohérence des valeurs retournées ;
- la variation des métriques en fonction de la charge système ;
- la stabilité du collecteur sur une exécution prolongée.

Les résultats observés confirment le bon fonctionnement du module de collecte des métriques. Cette phase est ainsi considérée comme validée et constitue une base fiable pour la suite de l'implémentation.

1.5 Conclusion de la phase

La phase de collecte des métriques système a permis de mettre en place un premier composant fonctionnel de l'agent de surveillance. Ce composant fournit des données fiables et horodatées sur l'état global de la machine, et pourra être directement réutilisé lors de l'implémentation des mécanismes de communication réseau et de gestion des alertes.

2 Phase 1.2 : Envoi des métriques via UDP

Cette phase a pour objectif d'assurer la transmission périodique des métriques collectées par les agents de surveillance vers le serveur central. La communication repose sur l'utilisation du protocole UDP, choisi pour sa légèreté et sa faible latence, particulièrement adaptées à l'envoi fréquent de données de télémétrie.

2.1 Objectif

L'objectif principal de cette phase est de permettre à un agent de surveillance d'envoyer régulièrement des échantillons de métriques système au serveur central, sans établir de connexion persistante et sans bloquer l'exécution de l'agent.

2.2 Choix du protocole UDP

Le protocole UDP a été retenu pour l'envoi des métriques périodiques pour les raisons suivantes :

- faible surcharge réseau ;
- absence de mécanisme de connexion, réduisant la latence ;
- tolérance à la perte occasionnelle de paquets, les métriques étant envoyées de manière répétée ;
- simplicité de mise en œuvre.

Les métriques envoyées via UDP ne constituent pas des événements critiques, contrairement aux alertes, ce qui rend ce choix pertinent dans le contexte du projet.

2.3 Format des données

Les métriques sont sérialisées au format **JSON**, afin de garantir une structure claire, lisible et interopérable. La sérialisation est réalisée manuellement, sans dépendance externe, conformément à l'objectif d'implémentation en Java pur.

Chaque message UDP contient un échantillon de métriques incluant :

- un horodatage ;
- l'utilisation du processeur ;
- l'utilisation de la mémoire ;
- l'utilisation de l'espace disque.

2.4 Implémentation côté agent

Un module dédié à l'envoi des métriques via UDP a été implémenté au sein de l'agent de surveillance. Ce module est responsable de la conversion des métriques en JSON et de leur transmission vers le serveur central à intervalles réguliers.

L'envoi est réalisé à l'aide d'un socket UDP ouvert temporairement pour chaque transmission, garantissant une implémentation simple et robuste.

2.5 Implémentation côté serveur

Afin de valider le bon fonctionnement de la communication UDP, un serveur UDP minimal a été développé. Ce serveur est chargé de recevoir les messages envoyés par les agents et d'afficher leur contenu dans la console.

Ce composant de test permet de vérifier la réception correcte des messages, leur format et leur cohérence, avant l'intégration dans le serveur central définitif.

2.6 Tests et validation

Les tests ont été réalisés en exécutant simultanément :

- un agent de surveillance envoyant périodiquement des métriques ;
- un serveur UDP en écoute sur le port dédié.

Les résultats observés confirment la bonne transmission des données, la cohérence des messages JSON et la stabilité de la communication dans le temps.

Cette phase est ainsi considérée comme validée et constitue une étape essentielle vers l'implémentation complète du serveur central.

2.7 Conclusion de la phase

La phase d'envoi des métriques via UDP a permis de mettre en place un mécanisme de communication efficace entre les agents et le serveur. Elle fournit une base solide pour l'intégration ultérieure du traitement, du stockage et de l'analyse des données au niveau du serveur central.

3 Phase 1.3 : Gestion des seuils et alertes critiques via TCP

Cette phase vise à mettre en place un mécanisme de détection et de notification des situations critiques au niveau des agents de surveillance. Contrairement aux métriques périodiques envoyées via UDP, les alertes critiques constituent des événements importants nécessitant une transmission fiable et immédiate vers le serveur central.

3.1 Objectif

L'objectif principal de cette phase est de permettre aux agents de :

- évaluer localement les métriques collectées par rapport à des seuils prédéfinis ;
- détecter les situations critiques en temps réel ;
- transmettre immédiatement une alerte fiable au serveur central.

La détection des alertes est réalisée localement afin de réduire la latence et d'éviter toute dépendance au serveur central pour l'identification des situations critiques.

3.2 Définition des seuils

Les seuils d'alerte sont définis de manière statique au niveau de l'agent. Deux niveaux de seuils peuvent être envisagés :

- un niveau *Warning*, indiquant une situation dégradée ;
- un niveau *Critical*, indiquant une situation nécessitant une réaction immédiate.

Dans le cadre de cette phase, seuls les seuils critiques sont pris en compte pour le déclenchement des alertes, afin de limiter le volume de notifications transmises au serveur.

Les seuils sont centralisés dans une classe dédiée, facilitant leur modification et leur évolution ultérieure.

3.3 Modélisation des alertes

Une alerte est modélisée comme un événement immuable comprenant :

- un horodatage ;
- le type de métrique concernée ;
- la valeur mesurée ;
- le niveau de criticité.

Les alertes sont sérialisées au format JSON afin de garantir une structure claire, lisible et interopérable.

3.4 Choix du protocole TCP

Le protocole TCP a été retenu pour la transmission des alertes critiques pour les raisons suivantes :

- garantie de livraison des messages ;
- ordre de transmission préservé ;
- fiabilité adaptée à des événements critiques ;
- simplicité d'implémentation côté agent et serveur.

Chaque alerte est transmise via une connexion TCP dédiée, ouverte le temps de l'envoi puis immédiatement fermée.

3.5 Implémentation côté agent

Le mécanisme d'alerte côté agent repose sur trois composants principaux :

- un module d'évaluation des seuils, chargé de comparer les métriques collectées aux seuils critiques ;
- un modèle représentant les alertes ;
- un client TCP chargé de transmettre les alertes au serveur central.

À chaque cycle de collecte, les métriques sont évaluées. En cas de dépassement d'un seuil critique, une ou plusieurs alertes sont générées et envoyées immédiatement au serveur.

3.6 Implémentation côté serveur (test)

Afin de valider la transmission des alertes, un serveur TCP minimal a été implémenté à des fins de test. Ce serveur est chargé d'écouter sur un port dédié, de recevoir les alertes envoyées par les agents et d'afficher leur contenu.

Ce composant de test permet de vérifier la fiabilité de la communication TCP et la cohérence des messages reçus avant l'intégration dans le serveur central définitif.

3.7 Guide de test et validation

Cette section décrit la procédure de test permettant de valider le bon fonctionnement du mécanisme d'alertes critiques.

3.7.1 Préparation de l'environnement

Les étapes suivantes doivent être réalisées :

- compiler les classes de l'agent de surveillance ;
- compiler le serveur TCP de test ;
- s'assurer que le port TCP utilisé pour les alertes est libre.

3.7.2 Lancement du serveur TCP

Le serveur TCP de test est lancé en premier afin d'être prêt à recevoir les alertes :

```
javac TcpServerTest.java
java TcpServerTest
```

Le message indiquant que le serveur est en écoute confirme son bon démarrage.

3.7.3 Lancement de l'agent de surveillance

L'agent est ensuite lancé dans un second terminal :

```
javac agent/*.java
java agent.AgentAlertTest
```

L'agent commence alors à collecter les métriques et à évaluer les seuils définis.

3.7.4 Déclenchement d'une alerte

Afin de tester le mécanisme d'alerte, une situation de charge artificielle peut être créée sur la machine surveillée (par exemple en augmentant la charge CPU). Il est également possible d'abaisser temporairement les seuils critiques afin de provoquer rapidement une alerte.

3.7.5 Résultats attendus

Lorsqu'un seuil critique est dépassé :

- l'agent affiche l'envoi d'une alerte dans la console ;
- le serveur TCP affiche la réception de l'alerte au format JSON.

Ces résultats confirment le bon fonctionnement du mécanisme de détection et de transmission des alertes critiques.

3.8 Conclusion de la phase

La phase de gestion des alertes critiques via TCP a permis de mettre en place un mécanisme fiable de notification des situations critiques. Elle complète les fonctionnalités de l'agent de surveillance en ajoutant une capacité de réaction immédiate, essentielle pour un système de supervision distribué. Cette phase constitue une base solide pour l'intégration du serveur central définitif et la mise en place de la gestion globale des alertes.

4 Phase 2 : Serveur central de supervision

Cette phase correspond à la mise en œuvre du serveur central du système de supervision distribuée. Le serveur constitue le cœur de l'architecture : il est chargé de recevoir les métriques envoyées par les agents, de traiter les alertes critiques, et de maintenir l'état global des machines surveillées.

4.1 Rôle du serveur central

Le serveur central remplit les fonctions suivantes :

- réception des métriques système transmises périodiquement via le protocole UDP ;

- réception fiable des alertes critiques via le protocole TCP ;
- gestion de l'état des agents de surveillance ;
- préparation des données pour une exposition ultérieure via une API REST.

L'architecture du serveur est conçue de manière modulaire afin de garantir sa maintenabilité et son évolutivité.

—

4.2 Phase 2.1 : Réception des métriques via UDP

4.2.1 Objectif

L'objectif de cette sous-phase est de permettre au serveur central de recevoir un flux continu de métriques système envoyées par les agents de surveillance. Le protocole UDP est utilisé afin de minimiser la latence et d'éviter tout blocage lié à des mécanismes de retransmission.

4.2.2 Principe de fonctionnement

Le serveur ouvre un socket UDP sur un port dédié. À chaque réception d'un datagramme :

- le message JSON est décodé ;
- les métriques sont extraites ;
- les informations sont associées à l'agent émetteur ;
- la date de dernière activité de l'agent est mise à jour.

Les métriques sont stockées en mémoire afin de conserver l'état le plus récent de chaque agent.

4.2.3 Choix techniques

- Utilisation de `DatagramSocket` pour la communication UDP ;
- Parsing JSON minimaliste en Java pur ;
- Stockage des agents dans une `ConcurrentHashMap` afin de garantir la sûreté en environnement concurrent.

—

4.3 Phase 2.2 : Réception des alertes critiques via TCP

4.3.1 Objectif

Cette sous-phase vise à assurer la réception fiable des alertes critiques émises par les agents. Contrairement aux métriques périodiques, une alerte critique doit impérativement être transmise sans perte.

4.3.2 Principe de fonctionnement

Le serveur écoute sur un port TCP dédié aux alertes. Chaque connexion TCP correspond à l'envoi d'une alerte :

- le message JSON est lu depuis le flux d'entrée ;
- l'alerte est décodée et stockée ;
- l'état de l'agent concerné est immédiatement mis à jour.

Un modèle multi-thread est utilisé afin de permettre la gestion simultanée de plusieurs alertes.

4.3.3 Choix du protocole TCP

Le protocole TCP a été retenu pour les alertes pour les raisons suivantes :

- garantie de livraison des messages ;
- intégrité des données ;
- simplicité d'implémentation ;
- adéquation avec la criticité des événements transmis.

—

4.4 Phase 2.3 : Gestion de l'état des agents

4.4.1 Modèle d'état

Chaque agent est associé à un état logique représentant sa situation courante :

- **INIT** : agent jamais vu par le serveur ;
- **ONLINE** : métriques reçues récemment ;
- **ALERT** : alerte critique active ;
- **OFFLINE** : absence de métriques au-delà d'un délai prédéfini.

Ce modèle d'état est implémenté sous forme d'une machine à états simple.

4.4.2 Transitions d'état

Les transitions d'état sont déclenchées par les événements suivants :

- réception de métriques UDP : passage à l'état *ONLINE* ;
- réception d'une alerte TCP : passage à l'état *ALERT* ;
- dépassement d'un délai d'inactivité (TTL) : passage à l'état *OFFLINE*.

Un thread de surveillance périodique est chargé de détecter les agents inactifs et de mettre à jour leur état.

—

4.5 Architecture multi-thread du serveur

Le serveur central repose sur plusieurs threads indépendants :

- un thread dédié à la réception des métriques UDP ;
- un thread serveur TCP pour les alertes critiques ;
- un thread de surveillance des états des agents.

Cette architecture permet un fonctionnement non bloquant et assure la robustesse du serveur face à une charge croissante.

—

4.6 Guide de test et validation

4.6.1 Lancement du serveur

Le serveur central est lancé à partir du module serveur :

```
javac -encoding UTF-8 server/*.java
java server.ServerMain
```

Les messages de démarrage confirment l'activation des services UDP et TCP.

4.6.2 Tests fonctionnels

Les tests suivants permettent de valider le fonctionnement du serveur :

- réception continue des métriques UDP envoyées par un agent actif ;
- réception d'alertes critiques via TCP ;
- passage automatique d'un agent à l'état *OFFLINE* après arrêt de l'agent ;
- cohérence entre les métriques reçues, les alertes et l'état affiché.

4.6.3 Résultats observés

Les tests réalisés montrent que :

- les métriques sont correctement reçues et associées aux agents ;
- les alertes critiques sont transmises de manière fiable ;
- la machine à états reflète fidèlement la situation réelle des agents.

—

4.7 Conclusion de la phase

La phase 2 a permis de mettre en place un serveur central robuste et fonctionnel, capable de superviser plusieurs agents distants. Elle constitue une base solide pour l'extension du système, notamment l'ajout d'une API REST et d'une interface de visualisation.

5 Phase 3 : API REST et notifications temps réel

Cette phase vise à exposer les données internes du serveur central via une API REST, ainsi qu'à fournir un mécanisme de notification en temps réel permettant aux clients de recevoir automatiquement les mises à jour importantes. L'objectif est de permettre la consultation et la visualisation des informations de supervision sans couplage direct avec les mécanismes réseau UDP et TCP.

5.1 Objectifs de la phase

Les objectifs principaux de cette phase sont :

- exposer l'état des agents et les alertes via une API REST ;
- permettre la consultation détaillée d'un agent spécifique ;
- notifier les clients en temps réel lors des changements d'état ou de la réception d'alertes ;
- éviter toute approche de type polling côté client.

—

5.2 Architecture générale de l'API

L'API est implémentée à l'aide du serveur HTTP standard de Java (`HttpServer`), sans framework externe. Cette approche garantit une maîtrise complète du fonctionnement interne et une intégration fluide avec le serveur central existant.

L'API repose sur deux mécanismes complémentaires :

- une API REST classique pour la consultation à la demande ;
- des Server-Sent Events (SSE) pour la diffusion en temps réel des événements.

—

5.3 Phase 3.1 : Endpoint REST `/agents`

5.3.1 Description

L'endpoint `/agents` permet de consulter la liste des agents actuellement connus par le serveur central. Pour chaque agent, l'API retourne :

- l'identifiant de l'agent ;
- son état courant ;
- la date de dernière activité.

5.3.2 Format de réponse

La réponse est fournie au format JSON sous forme de liste.

5.3.3 Utilité

Cet endpoint fournit une vue synthétique de l'état global du système de supervision, adaptée à l'affichage d'un tableau de bord.

—

5.4 Phase 3.2 : Endpoint REST `/agents/{id}`

5.4.1 Description

L'endpoint `/agents/{id}` permet de consulter les informations détaillées d'un agent spécifique. Il retourne :

- l'état courant de l'agent ;
- la date de dernière communication ;
- les dernières métriques système reçues.

5.4.2 Gestion des erreurs

Si l'identifiant fourni ne correspond à aucun agent connu, l'API retourne une réponse HTTP 404 indiquant que l'agent est introuvable.

5.4.3 Intérêt fonctionnel

Cet endpoint est destiné à une consultation ciblée, par exemple lors de l'analyse d'un agent présentant un comportement anormal ou en état d'alerte.

—

5.5 Phase 3.3 : Endpoint REST `/alerts`

5.5.1 Description

L'endpoint `/alerts` permet de consulter l'ensemble des alertes critiques reçues par le serveur central. Chaque alerte contient :

- l'identifiant de l'agent concerné ;
- la métrique ayant déclenché l'alerte ;
- la valeur mesurée ;
- le niveau de criticité ;
- l'horodatage de l'événement.

5.5.2 Rôle

Cet endpoint permet d'analyser l'historique des incidents critiques et constitue une base pour la mise en place de mécanismes de notification ou de reporting.

—

5.6 Phase 3.4 : Notifications temps réel via Server-Sent Events

5.6.1 Principe des SSE

Les Server-Sent Events permettent au serveur d'envoyer automatiquement des messages vers le client via une connexion HTTP persistante. Contrairement à une approche par polling, le client n'a pas besoin d'interroger régulièrement le serveur pour obtenir les mises à jour.

5.6.2 Endpoint `/events`

L'endpoint `/events` maintient une connexion ouverte avec le client et diffuse les événements suivants :

- changement d'état d'un agent (ONLINE, ALERT, OFFLINE) ;
- réception d'une alerte critique.

Les événements sont envoyés au format SSE standard, comprenant un type d'événement et des données JSON associées.

5.6.3 Types d'événements

Deux types d'événements sont définis :

- `agent_state` : notification d'un changement d'état d'un agent ;
- `alert` : notification de la réception d'une alerte critique.

5.6.4 Avantages des SSE

Le recours aux Server-Sent Events présente plusieurs avantages :

- communication unidirectionnelle simple ;
- compatibilité native avec les navigateurs modernes ;
- faible surcharge réseau ;
- excellente adéquation avec les systèmes de supervision.

—

5.7 Guide de test et validation

5.7.1 Lancement du serveur

Le serveur central est lancé avec l'ensemble des services REST et SSE activés :

```
javac -encoding UTF-8 server/*.java
java server.ServerMain
```

5.7.2 Tests REST

Les endpoints REST peuvent être testés via un navigateur ou un outil HTTP :

- `/agents` : liste des agents ;
- `/agents/{id}` : détail d'un agent ;
- `/alerts` : liste des alertes.

5.7.3 Tests SSE

L'endpoint `/events` est accessible via un navigateur ou une application Web. Les événements sont reçus automatiquement lors :

- du démarrage ou de l'arrêt d'un agent ;
- du déclenchement d'une alerte critique.

Ces tests valident le bon fonctionnement du mécanisme de notification temps réel.

—

5.8 Conclusion de la phase

La phase 3 a permis d'enrichir le serveur central par une API REST complète et un système de notification temps réel. Cette combinaison offre une base solide pour le développement d'une interface de visualisation interactive et garantit une supervision efficace et réactive des agents distribués.