Vers une surveillance autonome : Optimisation DevOps et Machine Learning pour la détection d'anomalies en environnements Cloud

Arnaud Avocat-Gros[1]
[1] *Polytech Nice Sophia, Amadeus*

I. RÉSUMÉ

Dans le cadre de la migration des serveurs legacy vers le cloud Azure chez Amadeus, la mise en place d'une surveillance autonome et fiable des applications cloud devient essentielle pour garantir leur stabilité et réagir rapidement aux anomalies. Cette étude présente une solution de détection d'anomalies en temps réel pour des applications cloud distribuées, à travers une architecture conçue spécifiquement pour cette étude, utilisant Docker Compose, Prometheus, Grafana et Apache Storm. L'objectif est de simuler un environnement de production, avec des séries temporelles générées par l'application et des anomalies simulées, afin de tester l'efficacité de la détection d'anomalies dans un environnement cloud évolutif. Les données sont collectées en temps réel par Prometheus, qui scrute les métriques exposées par l'application et alerte en cas d'anomalies détectées. Apache Storm est utilisé pour exécuter des modèles de machine learning (DBSCAN, PersistAD, et LevelShift), chacun traitant les données pour identifier les anomalies. Les résultats montrent que les différentes approches de détection d'anomalies ont des performances variées en fonction des types d'anomalies à détecter. En particulier, le modèle DBSCAN a montré une excellente précision pour la détection d'anomalies complexes, tandis que PersistAD a eu des performances un peu moins bonnes, ce qui souligne que le choix du modèle doit être adapté au contexte d'application. Par ailleurs, les bandes de détection ont prouvé leur efficacité pour repérer les anomalies majeures, mais se montrent moins précises pour les variations régulières, illustrant ainsi que l'approche à privilégier dépendra des besoins spécifiques du système surveillé. L'architecture mise en place a permis de tester ces approches dans un cadre réaliste, bien qu'il s'agisse d'une simulation avec des métriques artificielles, ce qui n'a pas permis de reproduire toutes les typologies d'anomalies possibles dans un environnement réel. Cependant, les résultats obtenus montrent que cette architecture est scalable et adaptable, et qu'elle offre des pistes prometteuses pour la mise en place de solutions de détection d'anomalies dans des environnements cloud dynamiques. En résumé, cette étude souligne l'importance d'adapter les méthodes de détection d'anomalies en fonction des besoins spécifiques du système, tout en mettant en évidence l'efficacité de l'architecture proposée pour surveiller et maintenir des applications dans un environnement cloud.

CONTENTS

II. PRÉSENTATION DE LA SOCIÉTÉ AMADEUS

Amadeus est une entreprise technologique spécialisée dans les solutions de gestion pour l'industrie du voyage et du tourisme. Fondée en 1987, elle est aujourd'hui l'un des leaders mondiaux des services technologiques pour les

compagnies aériennes, les agences de voyage, les hôtels et d'autres acteurs du secteur. Son siège social est situé à Madrid, en Espagne, tandis que son principal centre de développement est basé à Sophia Antipolis, en France. L'entreprise conçoit et exploite des systèmes critiques pour les réservations, la gestion des vols et l'optimisation des opérations aériennes. Ses solutions couvrent un large éventail de services, notamment :

- Systèmes de réservation et de distribution : utilisés par les compagnies aériennes, les agences de voyage et les plateformes en ligne.
- Gestion des opérations aéroportuaires : optimisation des plannings des vols, contrôle des bagages et gestion des infrastructures aéroportuaires.

Amadeus joue un rôle clé dans la transformation digitale du secteur aérien en intégrant des technologies avancées comme le cloud computing, l'intelligence artificielle et l'optimisation des performances des infrastructures IT. Dans le cadre de cette évolution, l'entreprise adopte des stratégies DevOps et des solutions cloud pour améliorer l'efficacité, la scalabilité et la résilience de ses applications. Avec plus de 19 000 employés répartis dans plus de 190 pays, Amadeus continue d'innover pour répondre aux défis du secteur du voyage, en fournissant des solutions performantes et sécurisées à ses clients à travers le monde.

III. ÉTAT DE L'ART

Dans le domaine de la détection d'anomalies en temps réel dans des environnements distribués et complexes, plusieurs études ont proposé des approches intéressantes qui complètent et enrichissent notre travail. Par exemple, une étude [1] propose un modèle scalable pour la détection d'anomalies dans les systèmes cloud à grande échelle. Ce modèle utilise une analyse en composants principaux (PCA) robuste pour identifier les anomalies dans des environnements dynamiques, ce qui est similaire à notre propre approche d'analyse des séries temporelles et des anomalies dans les données. Bien que notre étude se concentre davantage sur l'utilisation de Storm pour la détection en temps réel, cette étude met en avant des concepts qui sont cruciaux pour la gestion des anomalies dans des systèmes aussi vastes et distribués que ceux du cloud. D'autre part, une autre étude [3] de aborde l'utilisation de la PCA robuste dans un contexte de détection d'intrusions où les données peuvent être non supervisées. Cette approche, qui utilise des composants principaux majeurs et mineurs pour distinguer les anomalies, partage des similarités avec notre utilisation de modèles basés sur l'analyse des données historiques pour détecter des comportements anormaux. Bien que nous nous concentrions sur des séries temporelles et des anomalies liées à des fluctuations de métriques, l'application de méthodes d'apprentissage non supervisé et de mesure de distance dans l'espace des composants principaux pourrait améliorer la détection dans notre propre étude. Enfin, l'étude [2] de traitant Apache Storm et l'identification des signes de vieillissement logiciel dans les systèmes de traitement d'événements en temps réel présente des résultats intéressants pour comprendre les problèmes de performance et d'anomalies dans un environnement de traitement de flux. Cette étude est particulièrement pertinente pour notre travail sur Apache Storm, car elle met en lumière les défis associés à l'exécution prolongée de systèmes distribués et aux anomalies qui peuvent survenir en raison du vieillissement logiciel, un phénomène que nous avons également observé dans nos propres expériences avec Storm. La gestion des ressources et l'optimisation de la mémoire, ainsi que la prévention des comportements anormaux dans ces systèmes, sont des sujets cruciaux à prendre en compte pour l'amélioration des performances en temps réel dans des environnements de grande échelle. Ainsi, ces travaux mettent en lumière différentes stratégies et défis dans le domaine de la détection d'anomalies en temps réel dans des environnements distribués et complexes, et leur intégration à notre propre étude permet de mieux comprendre les opportunités et les limitations des modèles actuels.

IV. CONTEXTE DE L'ÉTUDE

Amadeus, et plus particulièrement sa division Airline IT, est chargée de fournir des solutions clés en main aux compagnies aériennes, notamment des applications critiques telles que les systèmes de réservation de billets. Ces applications doivent être hautement disponibles, performantes et résilientes afin de garantir une expérience fluide aux utilisateurs finaux. Dans le cadre de l'évolution technologique d'Amadeus, une migration majeure des infrastructures informatiques est en cours : les serveurs historiques situés à Erding, en Allemagne, sont progressivement transférés vers le cloud Azure. Cette transition vise à améliorer la scalabilité, la résilience et la gestion des coûts des applications tout en tirant parti des capacités avancées du cloud. Cependant, cette migration introduit également de nouveaux défis en termes de monitoring et de gestion des incidents. Lors du passage vers le cloud, certaines applications peuvent adopter des comportements imprévus en raison de différences architecturales entre les infrastructures on-premise et Azure. Par conséquent, il est essentiel d'identifier rapidement les anomalies qui pourraient impacter la disponibilité ou la performance des services en production. Ce projet de recherche vise à explorer et à mettre en place des méthodes avancées de détection d'anomalies afin d'anticiper les problèmes en production. En combinant des approches DevOps classiques (basées sur Prometheus et Grafana) et des techniques de Machine Learning (via Apache Storm et des modèles spécialisés), l'objectif est d'optimiser la surveillance des applications et d'améliorer

leur résilience dans l'environnement cloud.

V. MÉTHODOLOGIE UTILISÉE

A. Contraintes de l'étude

Plusieurs contraintes ont dû être prises en compte pour réaliser cette étude. Tout d'abord, les différentes applications d'Amadeus utilisent des technologies de monitoring qui devaient être réutilisées pour cette étude : Prometheus pour l'agrégation des données et l'envoi des alertes, et Grafana pour la visualisation des données. De plus, un système scalable était nécessaire pour permettre de traiter un grand nombre de métriques et d'instances. Il était également essentiel que la détection d'anomalies soit effectuée en temps réel, afin de pouvoir alerter les bonnes personnes au bon moment. Ainsi, nous avons pu définir une architecture répondant à ces contraintes.

B. Architecture

Afin de répondre aux attentes de cette étude, nous avons conçu une architecture modulaire permettant de simuler le comportement d'une application dans le cloud Azure, exposant des métriques récupérées et stockées par Prometheus. Nous avons mis en place une solution avec Docker Compose comportant plusieurs conteneurs. Le premier est une application qui expose des métriques, visant à simuler le comportement d'une véritable application en production sur le cloud Azure. Cette application générera des données dont les valeurs varient en fonction du temps, telles qu'un flux élevé en journée qui baisse en soirée. Prometheus vient ensuite scrapper ces données de manière régulière (toutes les secondes) et les stocke sous la forme de séries temporelles. Il est ensuite possible de visualiser ces données et de créer des tableaux de bord montrant différentes informations à l'aide du conteneur Grafana. Parallèlement, nous avons mis en place une architecture utilisant Apache Storm, une solution permettant de traiter des flux de données en temps réel. Cette partie de l'architecture a pour objectif de détecter les anomalies en temps réel. Si cette partie détecte une anomalie, alors elle envoie une alerte à un conteneur Anomaly Exporter qui permet d'exposer ces alertes a Prometheus pour pouvoir les stocker et les visualiser.

1. *Application*

Comme mentionné précédemment, le conteneur application a pour objectif de simuler le comportement d'une application en exposant des données à Prometheus. Dans un contexte réel, une application peut générer une grande quantité de données, telles que la consommation du CPU, la consommation en mémoire et la consommation en réseau. De plus, chaque application peut exposer des données plus spécifiques et orientées métier. Par exemple, dans le cadre d'une application d'Amadeus, l'application pourrait exposer le nombre de vols consultés. Chaque donnée contient des labels permettant de rendre les informations plus précises et pertinentes.

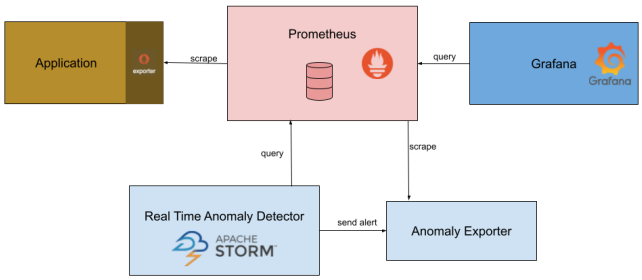

FIG. 1. Architecture globale de la solution

Pour notre expérience, nous avons décidé que l'application exposera une donnée spécifique, qui variera en fonction du temps. Cette donnée simulera une haute consommation pendant la journée et une consommation plus faible pendant la nuit. De plus, l'application introduira des anomalies de manière aléatoire afin de tester les différentes méthodes de détection d'anomalies mises en place dans notre système.

2. *Prometheus*

Prometheus est une solution de monitoring open-source largement utilisée pour collecter, stocker et analyser des métriques sous forme de séries temporelles. Il est conçu pour surveiller les systèmes distribués et les applications modernes, en offrant une collecte de données efficace à travers un modèle pull où Prometheus interroge périodiquement les services via des endpoints HTTP. Ces données sont stockées dans une base de données optimisée pour les séries temporelles et peuvent être interrogées à l'aide du langage de requête PromQL, permettant ainsi une analyse approfondie des performances et de l'état des systèmes.

En plus de la collecte de données, Prometheus permet de configurer des alertes basées sur des seuils ou des conditions spécifiques, afin de notifier les équipes en cas d'anomalies ou de dégradations des services. Son intégration avec Grafana permet de visualiser ces métriques de manière interactive à travers des dashboards. Prometheus est particulièrement adapté aux environnements Cloud et DevOps, où il aide les équipes à détecter rapidement les problèmes de performance et à maintenir une haute disponibilité des services en production.

Dans notre cas, Prometheus va récupérer les informations de l'application ainsi que les alertes exposées par le conteneur Anomaly Exporter. Prometheus permettra également à Grafana de visualiser les données en temps réel, tout en fournissant à Apache Storm les données nécessaires pour un traitement en temps réel afin de détecter les anomalies.

3. *Détecteur d'anomalies en temps réel*

Apache Storm est un système de traitement de flux en temps réel conçu pour traiter des données à grande échelle dans des systèmes distribués. Il permet de traiter des millions de messages par seconde, ce qui en fait une solution idéale pour les environnements où des données doivent être analysées ou traitées instantanément, telles que les systèmes de surveillance en temps réel, l'analyse de données en streaming ou la détection d'anomalies. Storm est conçu pour être extrêmement scalable et tolérant aux pannes, en répartissant le traitement des données sur plusieurs nœuds de manière transparente.

Dans un système distribué, Storm utilise un modèle basé sur des spouts (sources de données) et des bolts (unités de traitement). Les spouts émettent des données, tandis que les bolts traitent ces données en parallèle, ce qui permet un traitement rapide et efficace des informations. Ce modèle le rend particulièrement adapté pour les applications nécessitant un traitement en temps réel, notamment pour la détection d'anomalies ou le suivi de métriques en continu, où il est crucial de détecter des problèmes dès qu'ils surviennent. Grâce à sa capacité à s'adapter à des volumes de données importants et à garantir la disponibilité des données, Apache Storm est un outil puissant pour les architectures distribuées à grande échelle.

Le Real-Time Anomaly Detector sera composé d'un spout chargé de récupérer les données toutes les secondes en effectuant des requêtes vers Prometheus. Ce spout émettra ensuite ces données vers trois bolts différents, chacun ayant pour objectif de détecter la présence d'anomalies dans le jeu de données. Chaque bolt sera basé sur un algorithme différent : l'un utilisera DBSCAN, un autre emploiera PersistAD, et le dernier se basera sur LevelShift (détecteur de la bibliothèque ADTK).

Lorsqu'un des bolts détecte une anomalie, il transmettra la donnée correspondante au dernier bolt, qui sera chargé d'émettre une alerte vers le conteneur Anomaly Exporter. Ce système permettra de détecter des anomalies en temps réel, en utilisant des algorithmes de machine learning appliqués à de grands volumes de données dans un environnement distribué. Cette architecture garantit la scalabilité de la solution, assurant ainsi une détection d'anomalies efficace même à grande échelle et dans des contextes dynamiques.

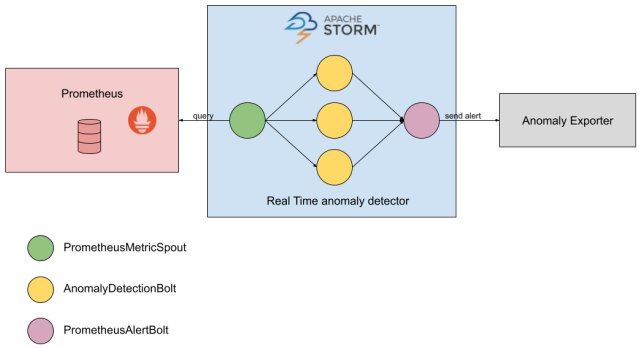

FIG. 2. Architecture du détecteur d'anomalies en temps réel

C. Les différents moyens testés pour détecter des anomalies

Pour cette étude, nous avons testé plusieurs méthodes de détection d'anomalies dans des données temporelles. Certaines approches se basaient sur des seuils simples, d'autres sur des analyses statistiques plus complexes, tandis que d'autres encore utilisaient des algorithmes de machine learning. Nous avons ainsi pu analyser différentes solutions, chacune présentant ses avantages et ses inconvénients.

1. *Détection basée sur des seuils*

La première méthode de détection d'anomalies consiste à définir des seuils constants ou variables en fonction de calculs statistiques. Ces approches présentent l'avantage d'être simples à implémenter, car elles peuvent être directement appliquées à travers des requêtes Prometheus, sans nécessiter l'utilisation de scripts externes pour effectuer des calculs supplémentaires. Cette simplicité facilite leur déploiement et leur intégration dans des systèmes existants, tout en offrant une solution efficace pour détecter des anomalies basées sur des critères bien définis.

a. Seuils constants La détection basée sur un seuil constant consiste à définir une valeur fixe au-delà de laquelle une donnée est considérée comme anormale. Cette méthode est simple à implémenter et permet de rapidement identifier les valeurs qui dépassent des limites préétablies. Elle est particulièrement efficace dans des contextes où les données suivent un comportement stable et prévisible, et où des anomalies évidentes se produisent lorsque les valeurs dépassent un seuil défini. Cependant, cette approche présente des limites dans des environnements dynamiques où les données fluctuent de manière naturelle, car elle ne prend pas en compte les variations temporelles ou contextuelles. Par conséquent, elle peut entraîner un grand nombre de fausses alertes ou

manquer des anomalies subtiles si le seuil est mal ajusté.

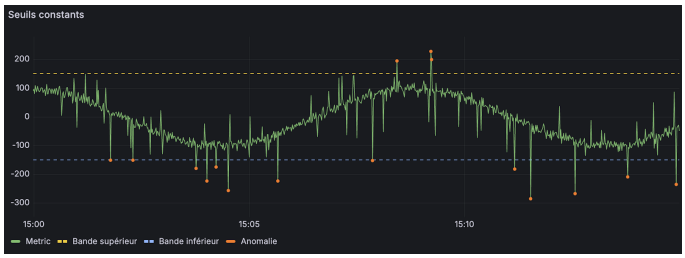

FIG. 3. Détection d'anomalies en fonction de seuils constants

b. Seuils calculés à partir de moyennes glissantes La détection basée sur des seuils calculés à partir de moyennes glissantes avec un offset permet de suivre l'évolution des données au fil du temps en ajustant dynamiquement les seuils de détection. Cette méthode consiste à calculer la moyenne des valeurs sur une fenêtre de temps glissante, puis à définir des seuils autour de cette moyenne en ajoutant ou en soustrayant un certain offset. Ce type de détection est plus flexible que les seuils constants, car il prend en compte les variations naturelles des données au cours du temps, permettant ainsi de mieux s'adapter aux comportements évolutifs. Toutefois, cette approche peut être sensible aux choix de la taille de la fenêtre et de l'offset, qui influencent directement la précision des détections. Si la fenêtre est trop grande, elle risque de lisser les anomalies importantes, tandis que si elle est trop petite, elle peut générer trop de fausses alertes.

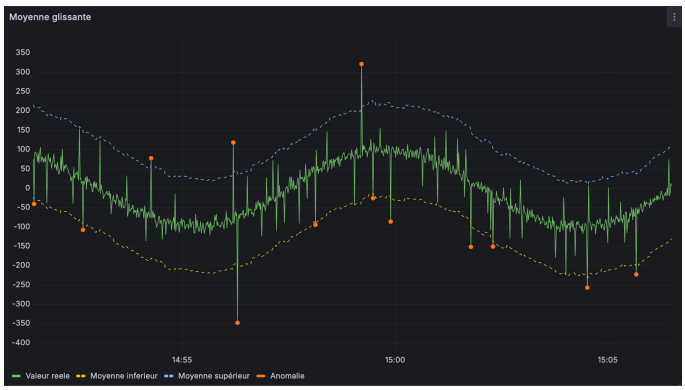

FIG. 4. Détection d'anomalies en fonction de seuils basés sur la moyenne glissante

c. Autres méthodes de calcul des seuils pour la détection d'anomalies Outre les seuils constants et les moyennes glissantes avec offset, il existe plusieurs autres techniques permettant de définir des seuils pour la détection d'anomalies dans des séries temporelles.

Une méthode couramment utilisée est celle des bandes de Bollinger, qui calcule des seuils dynamiques autour de la moyenne mobile d'une série de données. Ces bandes sont constituées de la moyenne mobile simple (SMA) de la série, entourée de deux bandes qui correspondent à un certain nombre d'écarts-types au-dessus et en-dessous de cette moyenne. Par exemple, dans le cas des bandes de Bollinger classiques, les seuils sont définis à deux écarts-types de la moyenne mobile. Les données situées en dehors de ces bandes sont considérées comme des anomalies, car elles s'écartent significativement de la tendance principale des données.

Cette méthode est particulièrement utile dans des environnements où les données présentent des fluctuations naturelles. Lorsque la volatilité des données augmente, les bandes s'élargissent pour tenir compte de l'augmentation des variations, et vice versa lorsque la volatilité est faible. Cela permet de détecter les anomalies sans avoir à fixer un seuil rigide, offrant ainsi une plus grande flexibilité. Cependant, il est important de noter que cette méthode peut échouer si les données suivent une tendance ou un cycle saisonnier fort, car les bandes risquent alors de ne pas capturer les anomalies contextuelles.

Une autre méthode consiste à utiliser des percentiles, où les seuils sont calculés en fonction de la distribution des données. Par exemple, en définissant un seuil au 95e percentile, toute valeur supérieure à ce seuil sera considérée comme anormale. Cette approche est particulièrement utile dans les cas où les données ne suivent pas une distribution normale, mais présentent des tendances extrêmes de manière sporadique.

Les approches basées sur les intervalles de confiance sont également couramment utilisées. Elles reposent sur des tests statistiques pour déterminer les plages dans lesquelles les valeurs sont censées se situer. Par exemple, en utilisant un intervalle de confiance à 95%, toute donnée en dehors de cet intervalle pourrait être vue comme une anomalie. Cependant, cette méthode suppose que les données suivent une certaine distribution, ce qui peut ne pas être le cas dans tous les environnements.

Ces différentes méthodes permettent de créer des seuils plus flexibles et adaptés aux spécificités des séries temporelles, mais chacune présente des avantages et des limites en fonction du type de données et du contexte dans lequel elles sont appliquées.

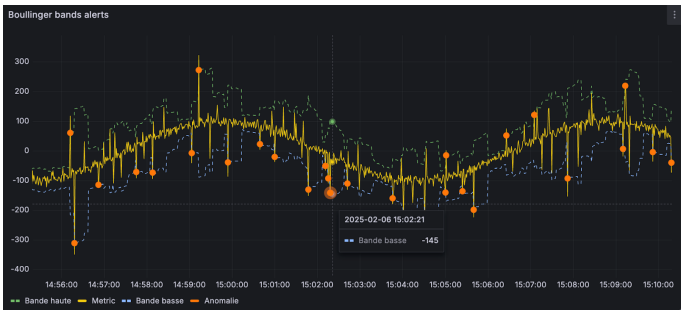

FIG. 5. Détection d'anomalies en fonction des bandes de Boullinger

2. *Détection basée sur des algorithmes de Machine Learning*

Une autre approche pour la détection d'anomalies dans des séries temporelles consiste à utiliser des algorithmes de machine learning. Ces approches pourraient être mieux adaptées dans certains cas, mais elles sont en revanche plus complexes à mettre en place. Ici, nous allons examiner deux algorithmes différents : DBSCAN et PersistAD, tout en gardant à l'esprit que d'autres solutions pourraient également être envisagées.

a. DBSCAN DBSCAN (Density-Based Spatial Clustering of Applications with Noise) est un algorithme de clustering qui regroupe les points densément connectés tout en identifiant les anomalies comme des points isolés. Bien qu'initialement conçu pour des données spatiales, il peut être appliqué aux séries temporelles en adaptant sa mesure de distance aux caractéristiques temporelles des données. Dans ce contexte, une approche consiste à utiliser DBSCAN sur des vecteurs temporels extraits via des fenêtres glissantes ou des embeddings temporels, permettant ainsi de détecter des régimes de comportement anormal. En analyse d'anomalies, DBSCAN est particulièrement efficace pour repérer des valeurs aberrantes, car il ne force pas tous les points à appartenir à un cluster. De plus, contrairement aux méthodes de clustering traditionnelles comme K-Means, il n'exige pas de spécifier un nombre fixe de clusters à l'avance, ce qui le rend robuste aux variations temporelles imprévues.

Dans notre cas, nous utilisons l'algorithme DBSCAN depuis un bolt Apache Storm pour détecter les anomalies en temps réel. Lorsqu'une anomalie est détectée, elle est transmise à un autre bolt, qui sera chargé de transmettre cette donnée à un container exposant les anomalies à Prometheus.

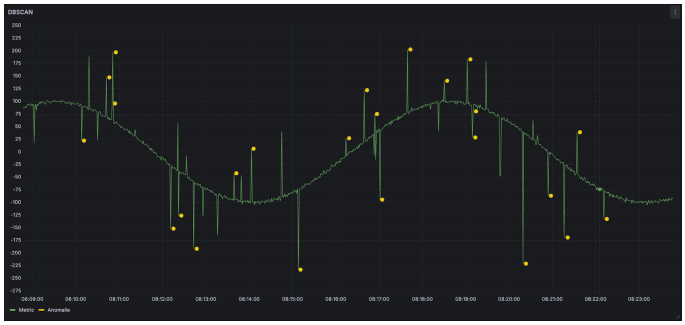

FIG. 6. Détection d'anomalies avec DBSCAN

b. PersistAD PersistAD (Persistent Anomaly Detection) est un algorithme conçu pour la détection d'anomalies dans les séries temporelles en tenant compte de la persistance des déviations. Contrairement aux méthodes qui identifient uniquement des points aberrants instantanés, PersistAD évalue la continuité des anomalies sur une période donnée, ce qui permet de mieux détecter des changements de régime ou des comportements anormaux prolongés. Il utilise des modèles d'apprentissage automatique pour capturer les tendances sous-jacentes et différencier les variations normales des véritables anomalies. En exploitant des fenêtres temporelles et des seuils dynamiques, PersistAD est particulièrement adapté aux environnements où les métriques évoluent de manière complexe, comme la surveillance des infrastructures cloud ou l'analyse des données IoT. Son approche permet ainsi d'éviter les faux positifs liés aux fluctuations temporaires et d'améliorer la robustesse des systèmes de détection d'anomalies.

Comme pour DBScan, PersistAD est appelé depuis un bolt Apache Storm qui fonctionne en parallèle de celui de DBScan.

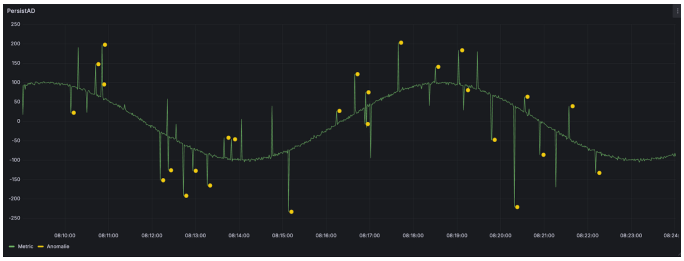

FIG. 7. Détection d'anomalies avec PersistAD

3. *Utilisation du windowing pour la détection en temps réel*

Le windowing joue un rôle essentiel dans l'application des algorithmes de détection d'anomalies comme DBSCAN et PersistAD. Chaque bolt est configuré avec une fenêtre de 60 secondes avançant par pas de 1 seconde, ce qui signifie que les données sont analysées sous forme de batches glissants plutôt qu'en points isolés. Cette approche permet de capturer la dynamique des métriques sur une période donnée, réduisant ainsi les faux positifs liés aux fluctuations instantanées. Pour DBSCAN, le windowing garantit que les points sont évalués dans un contexte temporel cohérent, en permettant la détection de groupes anormaux de valeurs sans avoir à conserver l'historique complet des données. De son côté, PersistAD bénéficie du windowing pour suivre l'évolution persistante des anomalies et ne pas réagir à de simples variations ponctuelles. En combinant cette gestion des fenêtres avec un traitement distribué en streaming, Storm assure une exécution efficace et en temps réel de la détection d'anomalies sur des flux de données continus.

Il est évident que nous pouvons ajuster la durée et le glissement des fenêtres pour améliorer les résultats, mais il est important de prendre en compte les contraintes liées à l'augmentation de la durée de la fenêtre. Tout d'abord, une fenêtre plus large entraîne une latence accrue, car les anomalies ne peuvent être détectées qu'après l'accumulation d'un nombre suffisant de données, retardant ainsi la réaction du système. Ensuite, cela augmente la consommation de mémoire et la charge de calcul, car

chaque bolt doit conserver un historique plus long des métriques avant d'effectuer ses analyses, ce qui peut poser problème dans un environnement aux ressources limitées. De plus, une fenêtre trop grande risque de diluer l'effet des anomalies à court terme, les rendant moins visibles si elles sont noyées dans une longue période de données normales. Enfin, cela peut également réduire la réactivité du modèle face à des changements rapides dans le comportement des métriques, ce qui est problématique pour des systèmes nécessitant une détection en temps réel. Ces points sont donc essentiels à prendre en compte, étant donné que nous devons concevoir une architecture capable de répondre en temps réel.

4. Tester la solution

La solution est entièrement disponible sur GitHub et un exemple de simulation est disponible ici.

VI. RÉSULTATS ET DISCUSSIONS

Nous avons pu mesurer la détection d'anomalies au fil du temps sur les différents modèles de machine learning que nous avons testés. Pour ce faire, nous avons créé une nouvelle donnée dans notre application, qui est incrémentée à chaque fois qu'une anomalie est détectée, afin d'avoir un point de comparaison. Ensuite, grâce à une requête Prometheus, nous avons agrégé le nombre d'anomalies détectées pour chaque modèle, puis exporté ces résultats dans un tableur pour analyse. Nous observons que les deux modèles ne parviennent pas à détecter l'intégralité des anomalies générées. En revanche, nous pouvons constater que le modèle DBSCAN présente une meilleure précision que PersistAD. Bien sûr, le choix des paramètres influe sur ces résultats, et il est possible que d'autres paramètres modifient les performances des modèles. Concernant la détection par bandes, nous n'avons pas pu agréger ces résultats avec ceux des autres modèles. Toutefois, nous avons observé une bonne précision sur des anomalies majeures (c'est-à-dire des données vraiment différentes de la normale). En revanche, lorsque nous utilisons des données qui varient de façon régulière, les modèles basés sur la détection par bandes sont moins précis que les modèles de machine learning. Il est à noter que les modèles de machine learning utilisant l'architecture de streaming avec Apache Storm présentent un certain délai, d'environ une seconde, lors de la détection des anomalies. Bien sûr, il convient de prendre du recul sur ces résultats : les métriques que nous avons utilisées ne sont pas réelles. Nous avons uniquement émis des anomalies sous forme de pics (spikes). Dans un cas réel, d'autres types d'anomalies pourraient être détectés. Par exemple, on pourrait envisager une élévation progressive de la donnée pendant un certain temps, ce qui ne correspond pas à un spike.

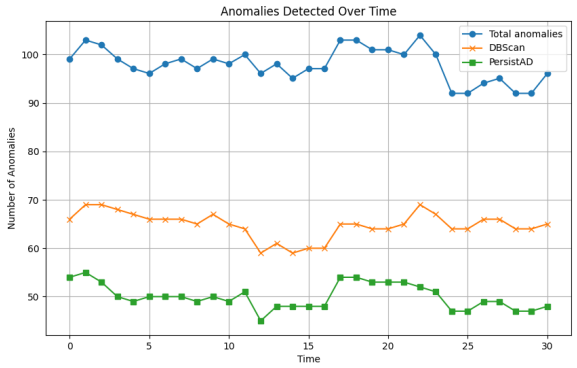

FIG. 8. Comparaison des différents modèles de ML utilisé pendant l'étude

VII. CONCLUSIONS ET PROCHAINS TRAVAUX

Dans cette étude, nous avons exploré diverses méthodes de détection d'anomalies en temps réel en utilisant des modèles de machine learning et des techniques de détection par bandes. Les résultats obtenus montrent que, bien que les deux approches présentent des avantages et des inconvénients, les modèles de machine learning, en particulier le modèle DBSCAN, offrent une meilleure précision pour la détection des anomalies, surtout dans des environnements où les anomalies sont significatives et distinctes des données normales. Cependant, la détection par bandes a montré une certaine efficacité pour repérer des anomalies importantes, bien qu'elle soit moins performante avec des données fluctuantes de manière régulière.

L'architecture de streaming basée sur Apache Storm a introduit une légère latence, mais celle-ci reste minimale et n'affecte pas de manière significative la réactivité du système dans des scénarios de détection en temps réel. Cela dit, la mise en place d'un système de streaming avec Apache Storm comporte une certaine complexité, notamment en raison de la gestion des flux de données en temps réel, de la coordination entre les différents composants et de la nécessité d'une infrastructure robuste. De plus, étant un système distribué, Apache Storm peut devenir coûteux à grande échelle, en termes de ressources matérielles et de gestion de l'infrastructure, ce qui constitue un défi pour les environnements nécessitant une évolutivité.

En dépit de ces défis, cette étude a mis en évidence les avantages d'une architecture de streaming pour la détection d'anomalies en temps réel. Les modèles de machine learning, bien que performants, nécessitent un ajustement précis des paramètres pour obtenir les meilleurs résultats, et l'architecture distribuée permet d'assurer une certaine résilience et scalabilité, bien que cela vienne avec un coût supplémentaire.

Pour les travaux futurs, il serait intéressant d'étendre

cette étude à la détection d'anomalies non seulement dans les séries temporelles, mais aussi dans les logs systèmes. L'analyse des logs pourrait permettre de détecter des anomalies de comportement, des erreurs ou des patterns inhabituels dans des systèmes complexes, et pourrait complémenter efficacement les détections basées sur les séries temporelles. Ce futur travail pourrait également se concentrer sur l'optimisation des modèles pour réduire la latence tout en améliorant la précision dans des environnements à grande échelle.

---

[1] Chunming Rong Bikash Agrawal, Tomasz Wiktorski. Adaptive real-time anomaly detection in cloud infrastructures. *Concurrency and Computation, Practice and Experience*, 2017.

[2] Stefano Russo Massimo Ficco, Roberto Pietrantuono. Aging-related performance anomalies in the apache storm stream processing system. *Future Generation Computer Systems*, 86, 2818.

[3] Kanoksri Sarinnapakorn1 LiWu Chang Mei-Ling Shyu1, Shu-Ching Chen. A novel anomaly detection scheme based on principal component classifier.