

Designing a Real-Time Monitoring System for the AWS Cloud: An Adaptive Dashboard-Based Approach with Prometheus and Grafana *

Abdou Wahidi BELLO^{1,3}, Abdoul Kamal ASSOUMA^{2,3,*}, Tahirou DJARA^{2,3}, Francky Ruben Bignon HOUENOU^{2,3}

¹ *Faculté des Sciences et Techniques*

² *Ecole Polytechnique d'Abomey-Calavi*

³ *Université d'Abomey-Calavi*

Abstract

In this paper, we implemented a comprehensive monitoring system for the AWS cloud environment. The developed architecture is based on a secure AWS infrastructure using Amazon VPC for network segmentation, EC2 instances for hosting services, and Amazon S3 for data storage. The monitoring system integrates Prometheus for metrics collection and storage, coupled with Grafana for visualization through interactive dashboards. The obtained performance results show average scraping times of 0.212 seconds and query latencies as low as 0.0021 seconds, enabling near real-time monitoring of over 1,279 metrics collected from 3 targets.

Anomaly detection, implemented using the SH-ESD statistical model, demonstrated an accuracy of 88.85% on a sample of 330 data points. The model correctly identified 231 normal values and detected 29 anomalies during stress testing on EC2 instances. The automated alert system, managed by Alertmanager, ensures instant email notifications when critical thresholds are exceeded. This result confirms the robustness of the developed solution, providing proactive monitoring of the cloud infrastructure with rapid detection and response capabilities to incidents, while maintaining scalability in line with operational needs.

Keywords

Cloud, AWS, Monitoring, Anomaly Detection

1. Introduction

Cloud computing is emerging as a major revolution, particularly in the delivery of digital services. This model is based on the offshoring of computing resources whether computing power, storage, or applications to remote servers hosted in data centers accessible via the Internet. This approach frees users from the traditional constraints of managing and maintaining local infrastructure, such as purchasing expensive hardware, updating software, and managing hardware failures. However, despite its many advantages, cloud computing is not immune to the challenges inherent in complex IT systems, particularly when it comes to performance, reliability, and resource management. With the massive migration of infrastructure to the cloud, traditional monitoring tools, designed for static local environments, often prove inadequate. Furthermore, poor configuration of modern tools dedicated to monitoring cloud infrastructure can exacerbate this situation. In this context, open-source tools like Prometheus, designed for collecting and storing metrics, and Grafana, an interactive visualization platform, have established themselves as benchmarks in the field of monitoring in general. However, their potential remains under exploited when it comes to

CITA 2025— Emerging Technologies and Sustainable Agriculture, 26-28 June 2025, Cotonou, Benin

*Corresponding author.

✉ solfath@yahoo.fr (A. W. BELLO); abdoulk1@hotmail.com (A. K. ASSOUMA); csm.djara@gmail.com (T. DJARA); rubenhoudenou@gmail.com (R. B. HOUENOU)

ORCID 0000-0002-8591-6610 (T. DJARA)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

monitoring specific cloud infrastructures, particularly on AWS. Faced with this observation, this article proposes: a monitoring system optimized for the AWS cloud, based on the integration of Prometheus and Grafana. Beyond this classic approach, we enrich this system by implementing an anomaly detection based on the SH-ESD algorithm applied to CPU usage in our cloud environment.

2. Literary review

2.1. Cloud computing

Cloud computing or simply cloud is a paradigm born from the evolution of the Internet and virtualization. According to NIST (National Institute of Standards and Technology) cloud computing is a model enabling ubiquitous, convenient, on-demand network access to a shared set of configurable computing resources (networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or interaction with the service provider [1] . The various computing resources provided by the cloud are called cloud services. Cloud computing is characterized by:

- 1) self -service on demand: access to a service by a user is automatic and does not require interaction with the service provider.
- 2) Broadband access A network provides access to services. Access is standardized by heterogeneous clients, whether thin or thick (mobile phones, personal computers, and workstations).
- 3) pooling, a provider pools the resources it wishes to make available to its users.
- 4) elasticity: resources can be provided and reallocated quickly and automatically according to user demand.
- 5) pay- as-you-go billing, resource usage is monitored, controlled and reported to users and providers

The main cloud services are SaaS, IaaS, PaaS

A. SaaS (Software as a Service)

SaaS is the provision by a software provider of a software application to be used and purchased via the internet [2] . It does not require any installation on customer servers or client devices. Examples of popular SaaS products are Gmail, Microsoft 365, Shopify, GitHub, Dropbox.

B. IaaS (Infrastructure as Service)

IaaS is a model that allows users to rent and/or purchase computing infrastructure that includes, depending on the provider, servers, storage, computing power, and networking. For example, Amazon EC2 offers consumers resources, including CPU, memory, operating system, and storage, to meet user demands. [3]

C. PaaS (Platform as a Service)

PaaS is a model for developing, running, and managing business applications without having to build and maintain the infrastructure that such software development processes typically require. It provides an environment in which developers can build and deploy applications without necessarily needing to know how much memory and how many processors their application will use. [3] .

The major existing cloud providers are Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), IBM Cloud and Alibaba Cloud [4] .

2.2. Cloud monitoring

Cloud monitoring can be described as a set of manual or automated practices, solutions, and processes that help assess, measure, evaluate, and manage a cloud configuration more efficiently [5] . Cloud service monitoring ensures that all cloud-based websites, servers, and networks are operating optimally while providing analytical insights into risks, vulnerabilities, or capacity

issues. It involves collecting, storing, correlating, displaying metrics, and alarming and responding [6] . It includes:

1. website monitoring
2. database monitoring
3. virtual machine monitoring
4. virtual network monitoring
5. security and compliance monitoring.

2.3. Anomaly detection

Anomaly detection is the process of identifying data that deviates from expected patterns in a dataset. It is widely used in fields such as system monitoring, fraud detection, predictive analytics, and many others [7] . We distinguish point anomalies (a single abnormal observation within normal data), contextual anomalies (data is abnormal in a specific context, a single abnormal observation within normal data), and collective anomalies (a group of points is abnormal, but taken individually, the points can be normal). Several algorithms are used for anomaly detection in data monitoring. We distinguish between data-based methods (labeled, unlabeled, semi-labeled) and time series.

Among the algorithms used, we have: isolation Forest (iForest) clustering (K- Means, DBSCAN) autoencoders for unlabeled data, i.e. data with no normal or abnormal labels. Among the algorithms based on labeled data, we distinguish Random Forests, Support Vector Machines (SVM), convolutional neural networks (CNN) or recurrent neural networks (RNN).

A time series is a chronological and ordered set of data that shows the evolution over time of a measurement or value over time. In the case of time series, anomalies are often sudden deviations that cause either an increase or a decrease in traffic. Among the algorithms usable, we have LSTM (Long Short-Term Memory), Prophet, SHESD (Seasonal Hybrid Extreme Studentized Deviate).

2.4. Related works

In recent years, there has been a lot of research on monitoring in a cloud environment. Researchers, in their quest for an effective solution to monitoring problems, have formulated many models, designed frameworks, used feasible algorithms and techniques to improve monitoring in a cloud environment. Farshchi et al. [8] described a regression-based analysis technique to establish the correlation between the activity logs of an operation and its impact on cloud resources. The results show a success rate of 92.3% for detecting 115 injected faults. Marques et al. [9] proposes a framework for a Kubernetes -based cloud, using Prometheus for metrics collection and Grafana for visualization. This system improves response time and reduces service degradation, while providing a comprehensive view of application status, allowing malfunctions to be detected and resource utilization to be optimized during workload variations.

Khandelwal et al. [10] worked on a cloud monitoring framework that focuses on metrics such as available bandwidth and round-trip time between pairs of instances. To create this framework, various bandwidth and latency estimation tools were combined like Spruce. The proposed framework exhibits temporary congestions in EC2 instances.

Hamamoto et al. [11] propose a cloud-based log analysis environment using Elasticsearch and Kibana, applied to Moodle (an application written in PHP). The implementation uses Docker containers and integrates Prometheus for metrics collection and resource monitoring.

The study by Giamattei et al. [12] presents a systematic overview of monitoring tools for microservices and DevOps, examining 71 tools.

The study by Pragathi et al. [13] explores the implementation of a robust monitoring solution using Prometheus, Grafana, and Node Exporter in a Kubernetes environment. It addresses the challenges related to infrastructure monitoring and proposes a methodology to overcome them.

Jani 's article [14] describes the implementation of a unified monitoring system for microservices architectures, highlighting the advantages of Prometheus and Grafana for scalability.

Pedchenko et al. [4] present a comparative study of the main cloud providers. It appears that AWS is the most used cloud provider with a share of almost 50% of the current market in 2020.

2.5. Analysis of existing work

Farshchi et al. [8] demonstrate that regression analysis can identify defects in cloud resource monitoring with a high success rate, highlighting the need for more powerful tools. Similarly, the paper by Khandelwal et al. [10] reveals congestion and latency issues in monitoring EC2 instances. These studies show that despite AWS's monitoring capabilities (via CloudWatch), problems persist, particularly in terms of granularity and responsiveness. Alazani et al. [15] also show that CloudWatch does not allow monitoring of multi-cloud infrastructures and extensive customization of dashboards.

Marques et al. [9] and the study by Pragathi et al. [13] highlight the integration of Prometheus for metrics collection and Grafana for visualization in Kubernetes -based environments . These works highlight that this combination allows to improve the response time, to optimize the use of resources and to quickly detect malfunctions. The study of Giamattei et al. [12] emphasizes the effectiveness of Prometheus for the collection of metrics thanks to its HTTP-based scraping model, its great flexibility in the aggregation and analysis of data thanks to its multidimensional data model, its powerful query language (PromQL) and its proactive alerting system. This capacity is particularly relevant for an AWS environment, where services and resources are numerous and diversified. Yash Jani [14] describes the implementation of a unified monitoring system for microservices architectures, which allows real-time monitoring of application status and optimization of scalability. This framework can serve as a model for designing an AWS-specific system, capable of overcoming the limitations of CloudWatch.

2.6. Prometheus and Grafana

Prometheus is a powerful open-source monitoring system designed to observe the status of applications and infrastructure. It operates using a pull-based architecture, where the Prometheus server collects metric data from predefined targets at regular intervals. These metrics, expressed as time-series data, are gathered from sources like HTTP endpoints and stored in a dedicated time-series database (TSDB).

The system allows querying data using PromQL, Prometheus's query language, through an integrated HTTP server interface. Additional components include exporters, which expose metrics from various services (e.g., databases, web servers), and the Push gateway, which temporarily holds data from short-lived jobs. Service discovery helps dynamically detect targets, especially in environments like Kubernetes. Prometheus also supports alerting by generating alerts based on user-defined rules and forwarding them to Alertmanager, which organizes and routes notifications through channels such as email or PagerDuty.

Grafana, often used alongside Prometheus, is an open-source platform for creating interactive dashboards and visualizing data from multiple sources. It allows users to explore metrics, monitor trends, and spot anomalies through customizable visual elements like graphs and charts. Grafana also features alerting mechanisms that can notify users via various communication platforms like Slack or Microsoft Teams. With its flexible dashboard customization and plugin ecosystem, Grafana

supports real-time collaboration and integrates well with other tools to enhance infrastructure observability.

3. Materials and methods

3.1. Setting up a virtual network

Here we set up the entire architecture of our network, namely our subnets, our internet gateways, the routing tables using certain AWS services such as:

IAM: For managing roles and permissions across AWS services. This service allows us to implement our least-access policy to grant only the permissions strictly necessary to perform their tasks, and nothing more.

Table 1
IAM User Policies

Policies	Roles
AmazonEC2FullAccessfor	the configuration of instances EC2
AmazonS3FullAccess	for configuring S3 buckets
EC2InstanceConnect	to be able to connect to EC2 instances
ResourceGroupsandTagEditorFullAccess	facilitate the organization and management of resources
CloudWatchFullAccess	retrieve metrics and logs from services

VPC: This service allows us to create an isolated virtual network in AWS. We define our network's address range, routing tables, access rules, and subnets. The following figure shows the structure of our architecture

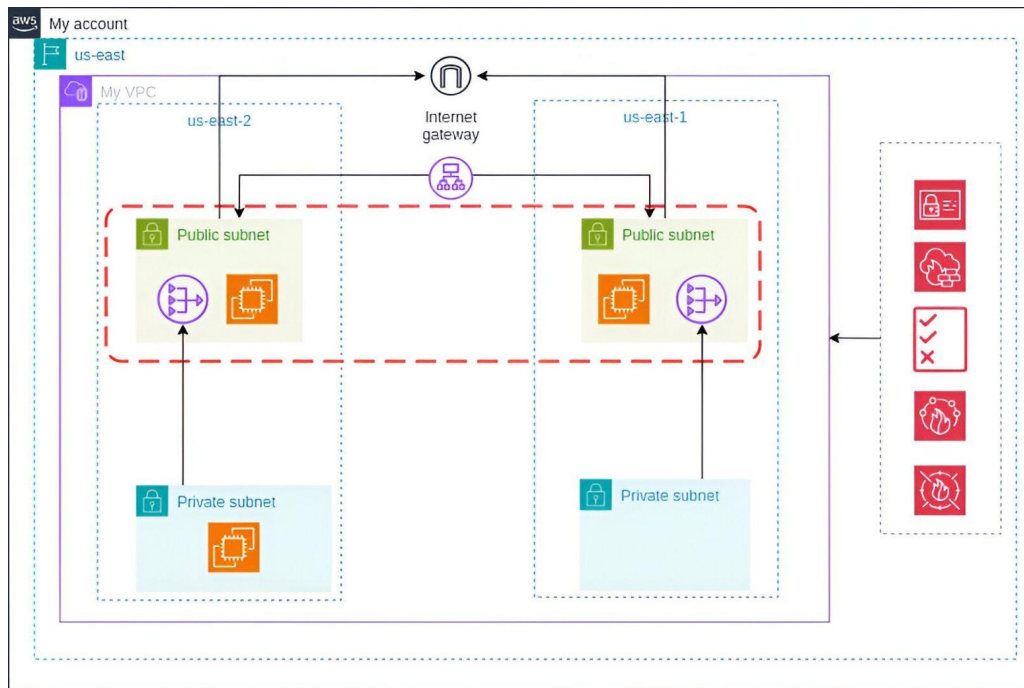


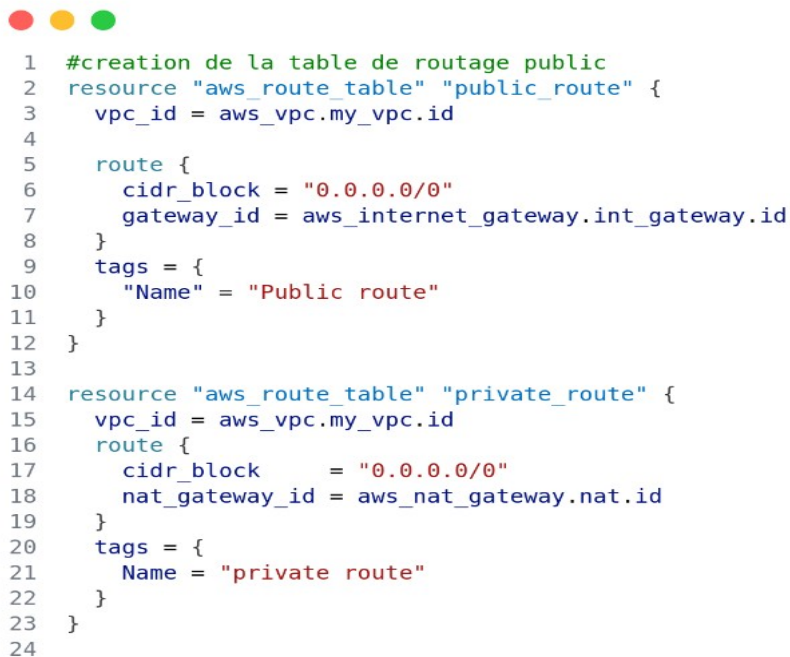
Figure 1: Network architecture

All of this infrastructure is defined and deployed using Terraform, an Infrastructure as Code (IaC) tool that allows us to manage our configuration in a declarative, versionable, and reproducible way. Using Terraform, we described the desired state of our infrastructure in configuration files and automated the deployment of resources.

Our VPC is configured with a private IP address range ensuring logical separation from other AWS users. At the moment of physical networks, we can configure routing tables, internet gateways, firewalls for our network. For our network, we define a range of IP addresses in our private cloud with the CIDR (Classless Inter-Domain Routing) block. Our range is 10.0.0.0/16 allowing to use all IP addresses ranging from 10.0.0.0 to 10.0.255.255 for a total number of 65,536 IP addresses. In our network, we have two types of subnets, public network subnets which host resources accessible from the internet, while private subnets contain protected internal resources, without direct access from the outside. Each subnet has its own routing table.

Terraform modules, allowing us to maintain a consistent configuration and quickly deploy identical environments if needed.

We placed our subnets in the us-east-2a and us-east-2b zones respectively. For the private subnets, a NAT gateway is used to allow the instances to access the internet for updates and other downloads while remaining inaccessible from the outside.



```

1  #creation de la table de routage public
2  resource "aws_route_table" "public_route" {
3      vpc_id = aws_vpc.my_vpc.id
4
5      route {
6          cidr_block = "0.0.0.0/0"
7          gateway_id = aws_internet_gateway.int_gateway.id
8      }
9      tags = {
10         "Name" = "Public route"
11     }
12 }
13
14 resource "aws_route_table" "private_route" {
15     vpc_id = aws_vpc.my_vpc.id
16     route {
17         cidr_block      = "0.0.0.0/0"
18         nat_gateway_id = aws_nat_gateway.nat.id
19     }
20     tags = {
21         Name = "private route"
22     }
23 }
24

```

Figure 2: Subnet configuration

In addition to the VPC and route tables, we configured two security mechanisms: an AWS Network ACL (NACL) and an AWS Security Group. The Network ACL allows us to define which types of traffic are allowed or blocked for instances located in the subnets. The public subnets allow inbound connections on port 80 for HTTP and 443 for HTTP. Port 22 allows SSH connections to the instances.

EC2. All outbound connections are allowed. For private subnets, outbound requests are blocked via the NAT gateway. The Security Group is a set of firewall rules applied directly to EC2 instances, unlike NACLs, which apply to entire subnets.

3.2. Setting up servers

In our network, we deploy a web server and a file server to serve as services for our network. For the web server, we used

EC2: This service allows us to manage and launch virtual machine instances. For our server, we chose a Linux Ubuntu 24.04 machine to run our web server. On this machine, we installed:

Node.js: JavaScript runtime environment with the Express JS framework for developing our API

PostgreSQL: Relational database management system.



```

1  resource "aws_instance" "web_serveur_a" {
2      ami             = var.ami
3      instance_type   = "t2.micro"
4      subnet_id       = var.subnet_public_a
5      key_name         = "serverweb"
6      connection {
7          type         = "ssh"
8          host         = self.public_ip
9          user         = "ubuntu"
10         private_key  = file("~/ssh/serverweb.pem")
11     }
12
13     provisioner "file" {
14         source       = "/home/ruben/node.sh"
15         destination = "/home/ubuntu/node.sh"
16     }
17
18     provisioner "remote-exec" {
19         inline = [
20             "chmod +x /home/ubuntu/node.sh",
21             "/home/ubuntu/node.sh", ]
22     }
23
24     associate_public_ip_address = true
25     security_groups             = [var.sec_group]
26 }
27

```

Figure 3: EC2 instance configuration

The figure above shows the configuration of our EC2 instance. With the node.sh script, we provision our machine directly.

S 3: Simple Storage Service (Amazon S3) is a data storage service from AWS that offers best-in-class scalability, data availability, security, and performance. It allowed us to set up our file server.

To facilitate the configuration and reuse of our infrastructure under AWS, we used Terraform, which is an infrastructure-as-code tool. It allows us to code our entire cloud configuration in computer language (HCL) to manage and deploy the cloud resources required for our project in an automated and structured manner.

3.3. Implementation of the monitoring system

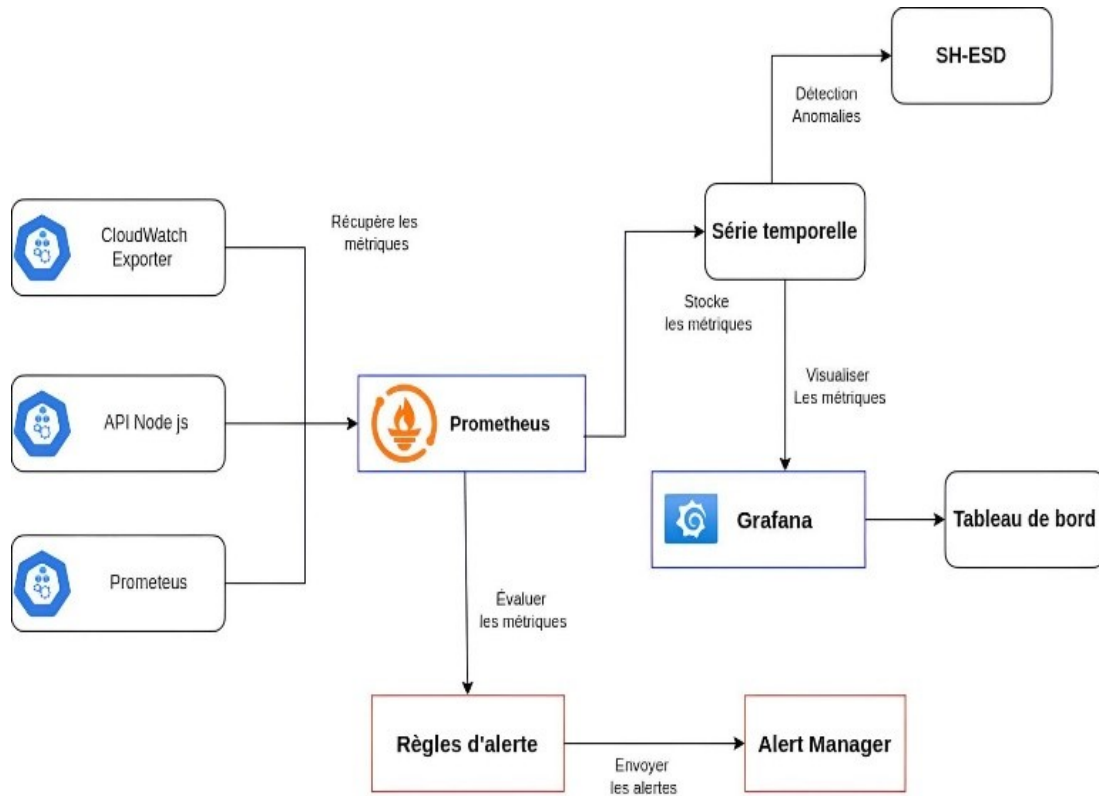


Figure 4: monitoring system

1 collecting metrics

Step monitoring pipeline. First, CloudWatch automatically captures performance data from our various cloud resources. Then, we use the CloudWatch Exporter for Prometheus, which extracts these metrics from CloudWatch, converts them to a Prometheus-compatible format, and then transmits them to our Prometheus server.

Prometheus server is configured to scrape metrics from three main sources:

- CloudWatch exporter mentioned above
- Our web server (via a dedicated exporter)
- Prometheus itself (self-monitoring)

This entire monitoring infrastructure is hosted on a dedicated EC2 instance within our network, guaranteeing both security and performance for our monitoring system.



```

1  global:
2    scrape_interval: 5s
3    evaluation_interval: 15s
4
5  scrape_configs:
6    - job_name: 'cloudwatch'
7      static_configs:
8        - targets: ["$ip_public:9106"]
9    - job_name: 'prometheus'
10     static_configs:
11       - targets: ["$ip_public:9090"]
12    - job_name: 'nodejs'
13     static_configs:
14       - targets: ["$ip_web_server_1:8000"]
15  alerting:
16    alertmanagers:
17      - static_configs:
18        - targets:
19          - "$ip_public:9093"
20  rule_files:
21    - "/etc/prometheus/alert_rules.yml"
22

```

Figure 5: prometheus server configuration

From Figure 5, we have three major sections in our file:

global in this section, we configure the metrics recovery interval (scrape interval) and re-evaluation of the rules

scrape configs in this section, we define the monitoring targets. Each target is defined with a job name which is an identifier for the target and a static configs where the addresses of the targets to be monitored are specified as URLs.

alerting: we define our Alertmanager for alert management.

2 Creating dashboards

We build the various dashboards with Grafana using the metrics collected from Prometheus. For this step, we started by setting up a connection between Grafana and Prometheus by adding Prometheus as a data source in Grafana. This configuration allows Grafana to query the metrics collected by Prometheus in real time.

Once we established this connection, we were able to create custom visualizations that transformed our raw data into intuitive graphical representations. Each dashboard is designed to meet specific monitoring needs, with charts, gauges, and tables that present the most relevant information for our infrastructure.

PromQL (Prometheus Query Language) queries are used to extract and transform data before displaying it in Grafana. This approach gives us great flexibility in creating our visualizations, allowing for aggregations, rate calculations, and comparisons between different metrics.

3 Anomaly detection

For anomaly detection, we opted for the SH-ESD model. To do this, we first retrieved the available metrics on CPU usage via our Prometheus server . After separating these data into different time series, we applied the ESD algorithm to detect possible outliers. Finally, we plotted these metrics by marking the different outliers.

- Alert system

To be notified when a threshold is exceeded, we set up an alert system using Prometheus and Alert manager. Prometheus collects and analyzes metrics, and Alert manager, a separate component, handles the reception and routing of alerts based on the rules we defined. These alert rules were fully configured through Alert manager and sends notifications by email.

4. Results and discussions

4.1. Results

This subsection aims to assess the level of interest of the scientific community in this Our AWS Virtual Private Network has been successfully deployed, spanning an IP address range from 10.0.0.0 to 10.0.255.255 (65,536 IP addresses). The architecture includes four strategically distributed subnets: two private subnets and two public subnets. This configuration ensures effective separation between public and private resources while optimizing internet connectivity without exposing sensitive components.

The developed API works correctly on our EC2 instance, accessible locally via the address " <http://127.0.0.1:8000> ". Validation of the operation was carried out using Thunder Client, confirming the availability and responsiveness of the API from the outside via the network's public IP address (Figure 6).

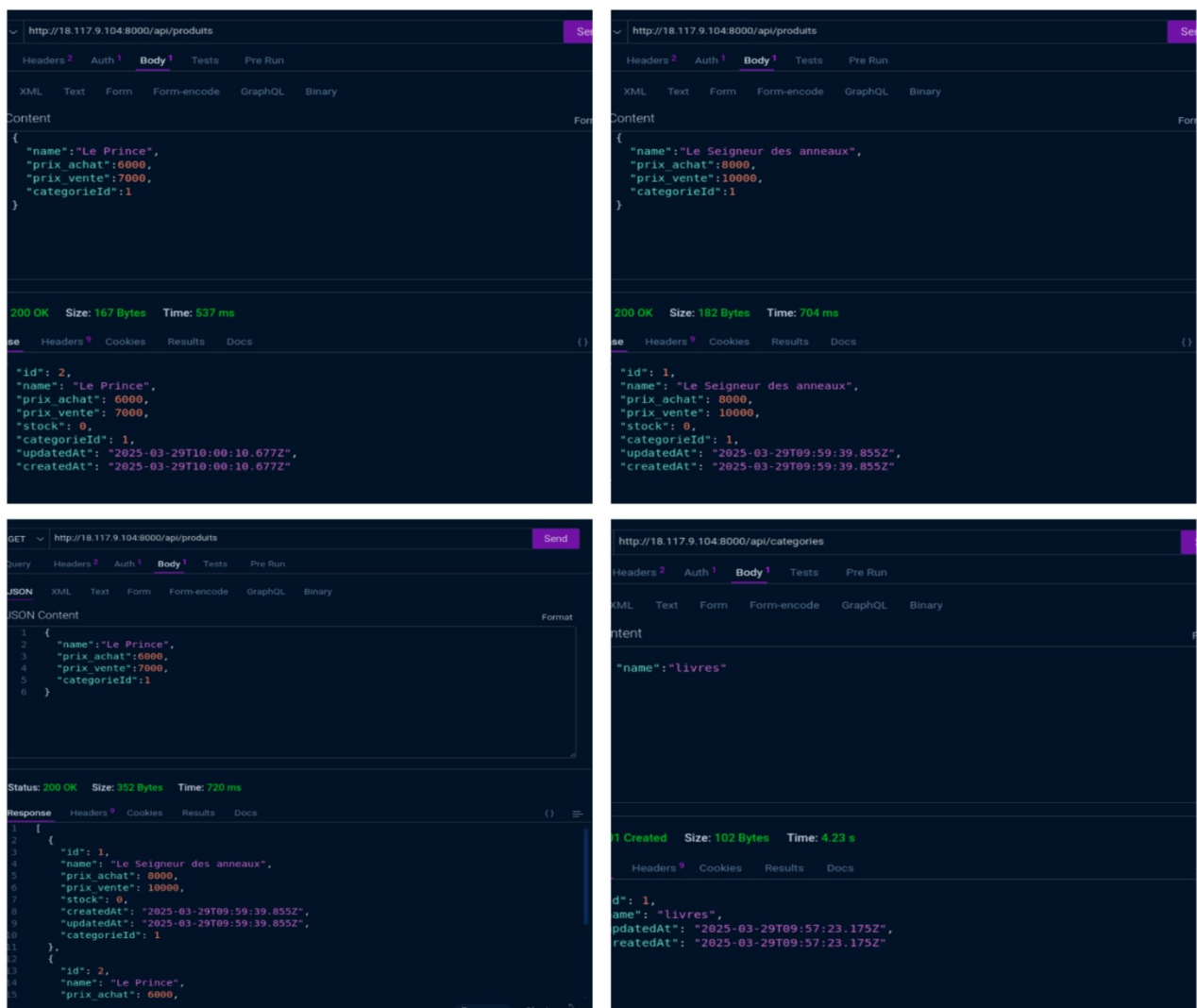


Figure 6: API requests

Analysis of the graphs highlights a significant evolution in the interest that scientists The performances measured over a one-hour period are presented in the following table:

Table 1

Performance measured with our solution

Criteria	Values with our solution
Scraping interval	5 seconds
Reassessment interval	15 seconds
scraping time	0.212 seconds
Request latency	0.0021 seconds
Total number of metrics retrieved	1279

scraping interval and re-evaluation interval help maintain a good balance between responsiveness and system load, avoiding unnecessary overload of our Prometheus server.

Although the number of targets is limited, these measurements can be extrapolated to assess the scalability of the system.

Our SH-ESD model was tested by simulating a one-time high CPU usage of an instance

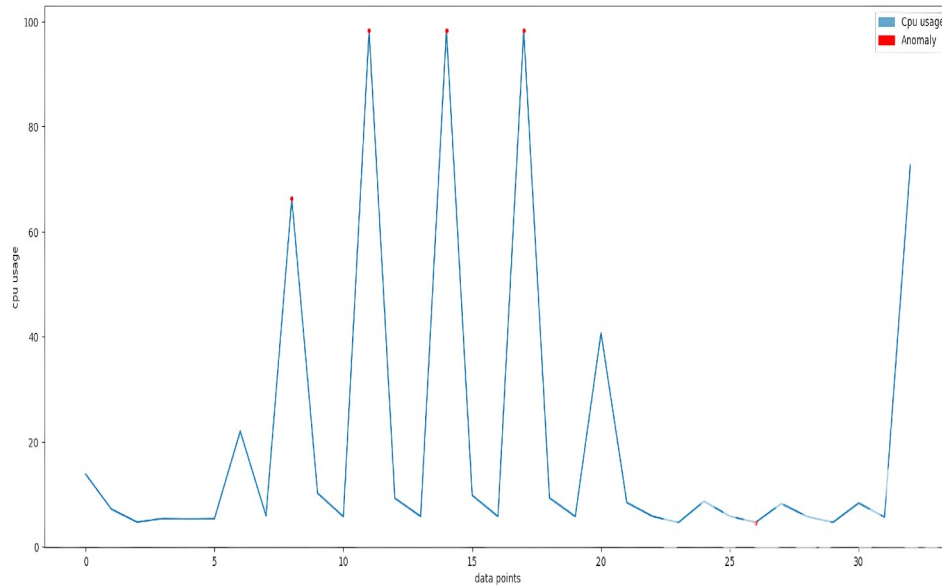


Figure 7: anomaly detection with our model

The results are presented in the following confusion matrix:

Table 3

Confusion matrix of the SH-ESD model

	Correct value predictions	Predictions of abnormal values
Correct values	231	29
Anomalies	30	40

This matrix gives us an overall accuracy of 88.85% over a total of 330 samples, demonstrating the effectiveness of our anomaly detection system.

Grafana dashboards, displaying metric variations in real time (Figure 12). These dashboards provide a clear graphical representation of performance and highlight trends, making it easier to anticipate potential incidents.



Figure 8: Dashboard for our web server

Figure 8 shows the dashboard created by our system for our web server. Figure 9 shows the dashboard created by our system for our EC2 instances.



Figure 9: Dashboard for our web server

4.2. Discussions

Despite the promising results, our solution has some limitations:

- ❖ Latency in metric collection: Using CloudWatch Exporter for Prometheus introduces a delay that may affect system responsiveness.
- ❖ Detection model reliability: The effectiveness of the SH-ESD model strongly depends on the quality and quantity of training data. The false positive rate could be reduced with additional adjustments.
- ❖ Scalability: While current results are promising, the system could benefit from integration with other AWS services like EKS for container monitoring or CloudWatch Logs Insights for deeper log analysis.

These limitations represent opportunities for improvement for future iterations of the monitoring system.

5. Conclusion and future work

During our work, we were able to implement a high-performance monitoring system for an AWS cloud environment, using the Prometheus and Grafana tools. We were thus able to address the challenges related to monitoring dematerialized infrastructures by proposing an approach integrating the collection, analysis and visualization of metrics essential to performance evaluation. The results obtained demonstrated the effectiveness of the implemented system, both in terms of reliability and adaptability.

However, some limitations remain, particularly regarding the improvement of the anomaly detection model's accuracy. Future work could therefore focus on exploring more advanced learning algorithms and integrating corrective action automation mechanisms. The work we have done could be complemented and continued by extending this solution to other cloud platforms such as Microsoft Azure and Google Cloud Platform, in order to offer a universal monitoring solution.

Declaration on Generative AI

During the preparation of this work, the authors used chatGPT-5 mini for the following activities: language refinement, grammar corrections, and occasional structural suggestions. After using this tool, the authors reviewed and edited all generated content as needed and take full responsibility for the publication's scientific integrity and content.

References

- [1] P. Mell and T. Grace , “The NIST Definition of Cloud Computing”.
- [2] Mr. Armbrust et al. , “Above the Clouds: A Berkeley View of Cloud Computing”.
- [3] A. Huth and J. Cebula , “The Basics of Cloud Computing”.
- [4] Y. Pedchenko , Y. Ivanchenko, I. Ivanchenko, I. Lozova , D. Jancarczyk , and P. Sawicki, “Analysis of modern cloud services to ensure cybersecurity”, *Procedia Comput . Sci .* , flight. 207, p. 110 -117, 2022, doi : 10.1016/j.procs.2022.09.043.
- [5] M. N. Birje and C. Bulla, “Cloud Monitoring System: A Review.”
- [6] Center for Post Graduate Studies, Visvesvaraya Technological University, Belagavi, Karnataka, India., MN Birje*, C. Bulla, and Dept. of CSE, KLE College of Engineering & Technology, Chikodi , Karnataka, India., “Cloud Monitoring System: Basics, Phases and Challenges”, *Int. J. Recent Technol. Eng. IJRTE* , vol. 8, no . 3, p. 4732 -4736, Sep 2019, doi : 10.35940/ijrte.C6857.098319.
- [7] W. Lu and AA Ghorbani , “Network Anomaly Detection Based on Wavelet Analysis”, *EURASIP J. Adv. Signal Process.* , flight. 2009, no . 1, p. 837601, Dec . 2008, doi : 10.1155/2009/837601.
- [8] M. Farshchi , J.-G. Schneider, I. Weber, and J. Grundy, “Experience report: Anomaly detection of cloud application operations using log and cloud metric correlation analysis,” in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)* , Gaithersbury , MD, USA: IEEE, Nov. 2015, p. 24 -34. doi : 10.1109/ISSRE.2015.7381796.
- [9] G. Marques, C. Senna, S. Sargento , L. Carvalho, L. Pereira, and R. Matos, “Proactive Resource Management for Cloud of Services Environments,” *Future Gener . Comput . Syst.* , flight. 150, p. 90 -102 , Jan. 2024, doi : 10.1016/j.future.2023.08.005.
- [10] H. Khandelwal, R. R. Kompella , and R. Ramasubramanian , “Cloud Monitoring Framework.”
- [11] N. Hamamoto, S. Yokoyama, A. Takefusa , and K. Aida, “ Implementation of Secured Log Analysis Environment for Moodle Using Virtual Cloud Provider Service,” *Procedia Comput . Sci.* , flight. 192, p. 3154 -3164, 2021, doi : 10.1016/j.procs.2021.09.088.
- [12] L. Giamattei et al. , “Monitoring Tools for DevOps and Microservices: A Systematic Gray Literature Review”, *J. Syst. Softw .* , flight. 208, p. 111906 , Feb. 2024, doi : 10.1016/j.jss.2023.111906.
- [13] P. Bc , H. Maddirala , and S. M, “Implementing an Effective Infrastructure Monitoring Solution with Prometheus and Grafana,” *Int. J. Comput . Appl.* , flight. 186, no . 38, p. 7 -15, None 2024.
- [14] Y. Jani, “Unified Monitoring for Microservices: Implementing Prometheus and Grafana for Scalable Solutions,” *J. Artif . Intellect . Mach. Learn. Data Sci.* , flight. 2, no . 1, p. 848 -852, March 2024, doi : 10.51219/JAIMLD/ yash-jani /206.
- [15] K. Alhamazani et al. , “An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art”, *Computing* , vol. 97, no . 4, p. 357 -377 , Apr. 2015, doi :10.1007/s00607-014-0398-5.