ICT-40-2020

H2020-ICT-2018-20

DataCloud

Enabling The Big Data Pipeline Lifecycle On The Computing Continuum

D5.4: DISTRIBUTED MONITORING AND RESOURCE ANALYSES

https://datacloudproject.eu/

Document Metadata

Work package	WP 5
Date	31/10/2023
Deliverable editor	Radu Prodan (AAU)
Version	0.1
Contributors	Narges Mehran (Partner Y)
Reviewers	Alexandre Ullises (MOG) Gøran Brekke Svaland (SINTEF)
Keywords	Resource analyses, Monitoring, Big Data pipelines, Computing Continuum
Dissemination Level	Public

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	01.04.2023	First table of content	Narges Mehran
V0.2	30.04.2023	Related work	Narges Mehran
V0.3	30.05.2023	Section 2 completed	Narges Mehran
V0.4	01.06.2023	Revision of the draft evaluation results	Narges Mehran
V0.5	01.06.2023	Sections 3, 5, 6 updating	Narges Mehran
V0.6	10.08.2023	Structuring the first draft	Radu Prodan

DISCLAIMER

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016835.

This document reflects only the authors' views and the Commission is not responsible for any use that may be made of the information it contains.

Co-funded by the Horizon 2020 Framework Programme of the European Union

EXECUTIVE SUMMARY

D5.4 presents a distributed community monitoring for efficient oversight of the large-scale decentralized pool of resources without a centralized monitoring infrastructure. The monitoring distributed among multiple devices collects local data from their neighbours and enables a highly scalable infrastructure. The locally analyzed monitoring data provides time-critical information and statistical profiling for data-aware provisioning and event detection. In this deliverable, we report our exploration and extension of the Prometheus and Netdata tools through their configurations, setups, metrics, queries, prediction and machine-learning-based model, time-window settings, etc., to monitor computing resources and networking channels.

Co-funded by the Horizon 2020 Framework Programme of the European Union

TABLE OF CONTENTS

Co-funded by the Horizon 2020
Framework Programme of the European Union

LIST OF FIGURES

Co-funded by the Horizon 2020 Framework Programme of the European Union

LIST OF TABLES

Co-funded by the Horizon 2020
Framework Programme of the European Union

ABBREVIATIONS

SPL	Step preference list
DPL	Device preference list

....

Co-funded by the Horizon 2020
Framework Programme of the European Union

1 INTRODUCTION

The recent shift towards the increasing number of user's applications in the Cloud-native infrastructure brings new scheduling, deployment, and orchestration challenges~\cite{joseph2020intma}, such as scaling out overloaded pipeline steps in response to increasing load.

In the DataCloud project~\cite{roman2022big}, Bosch developed one use case that is an ML-based application for welding quality monitoring consisting of four services: retrieving data from databases, slicing it into subsets, preparing, and storing it back in the database. In the Bosch use case, it is necessary to scale out the overloaded pipeline steps due to many requests exposing heavy traffic on the deployed applications.

To reduce the bottleneck on the computing infrastructure, ..

To detect the anomalous device in terms of their processing malfunctions ...

We apply ML models to predict ...

The deliverable has ... sections ...

Co-funded by the Horizon 2020
Framework Programme of the European Union

2 ARCHITECTURE

The architecture of ADA-PIPE consists of seven components, displayed in Figure 1: ADA-PIPE architecture..

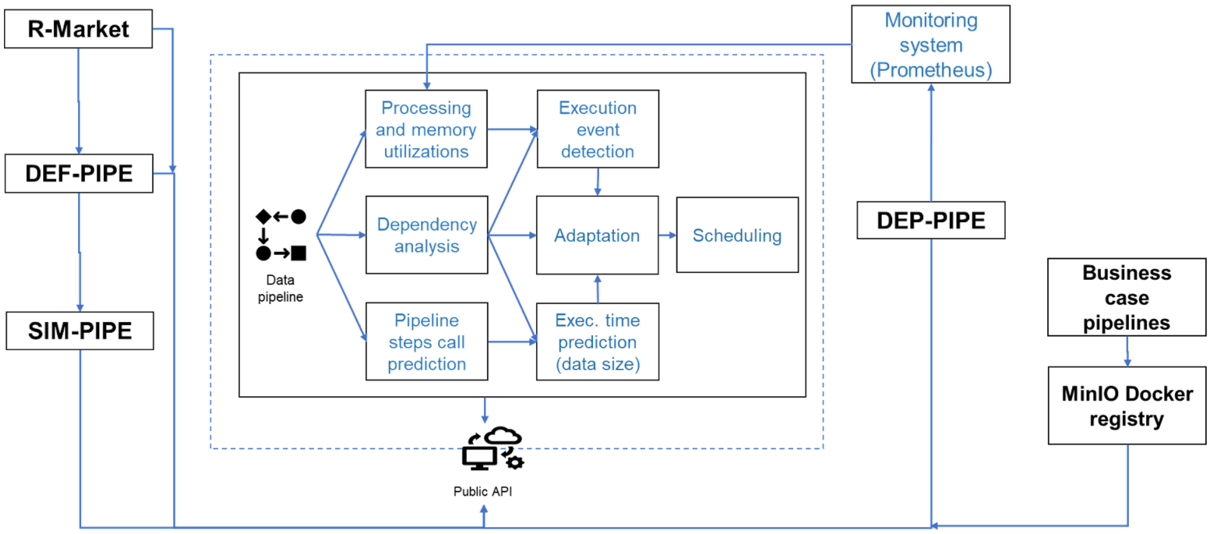


Figure 1: ADA-PIPE architecture.

Firstly and most importantly, ADA-PIPE pulls the data pipeline of the business case users (i.e., jot, mog, tellu, bosch, and ceramica) from the MinIO-based Docker registry.

1. **Dependency analysis** examines the input and output of all pipeline steps to identify parallelism. Based on the dataflow from the source to the sink of the pipeline, this component creates different chunks scheduled due to their requirements.
2. **Execution event detection** component analyses the monitoring data from the previous and current executions to identify anomalies that hinder the execution performance. This information improves the schedule and enables runtime adaptation.
3. **Processing and memory utilization** component conducts an analysis of the resource usage in the computing continuum and predicts the available resources for the user's data pipelines.
4. **Pipeline step call rate prediction** component applies the machine learning (ML) models and predicts the call rates of every data pipeline based on users' requests. More specifically, it predicts the call rates of the pipeline's steps based on their execution times.
5. **Execution time prediction** component estimates the processing time of the input Big data size by each pipeline's step. The prediction model applies a linear regression model taking less runtime overhead for scheduling.
6. **Adaptation** component applies re-scheduling or migration of the pipeline steps based on the analyzed monitoring received from the execution event detection. Both adaptation and scheduling interact with the deployment engine of DEP-PIPE, orchestrating the steps on available resources reserved by R-MARKET.
7. **Scheduling** component maps the pipeline steps to the resources using a matching theory algorithm applied to the step and resource preference lists in response to infrastructure drifts. ADA-PIPE receives information from the SIM-PIPE tool, providing the dry-run of the pipeline execution.

Co-funded by the Horizon 2020 Framework Programme of the European Union

ADA-PIPE includes a frontend with a REST API written in HTML and Bootstrap and a backend that exposes the Python Flask web application[1] to receive the results of the pipeline definition of the user and provides the pipeline's schedules. Specifically, ADA-PIPE utilizes the Flask swagger UI[2] to communicate with other tools in the DataCloud toolbox. In addition, the backend uses the Python NetworkX[3] library to analyze the dependencies of the pipeline steps. Therefore, ADA-PIPE exploits k-means clustering[4] for anomaly and event detection module[5]. To adapt the schedules, we rely on machine learning algorithms, including recurrent neural networks[6] and linear regression[7]. Finally, ADA-PIPE uses the capacity-, data-aware matching model[8] for the scheduling algorithm, which is based on the game theory principles.

1 https://github.com/DataCloud-project/ADA-PIPE/tree/main/frontend
2 https://pypi.org/project/flask-swagger-ui/
3 https://networkx.org/
4 https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
5 https://github.com/DataCloud-project/ADA-PIPE/tree/main/detect-anomalies
6 https://github.com/DataCloud-project/ADA-PIPE/blob/main/res-util-pred/lstm_models.py
7 https://scikit-learn.org/stable/auto_examples/linear_model/plot_ard.html
8 https://github.com/DataCloud-project/ADA-PIPE/tree/main/matching-scheduler

Co-funded by the Horizon 2020 Framework Programme of the European Union

3 ADA-PIPE'S MONITORING, ADAPTATION, AND SCHEDULING

This section illustrates the components related to the monitoring, followed by data preprocessing. Afterward, ADA-PIPE (re-)trains a k-means model on the preprocessed data to detect the anomalous execution of the pipeline's steps and adapt the initial schedules.

In detail, to record the pipeline executions on the computing continuum, the Prometheus monitoring system imports the NetData metrics, such as processor and memory utilization, along with the network bandwidth usage and the runtime of pipeline steps.

Afterward, ADA-PIPE trains an ML-based k-means model on the monitoring data. Furthermore, the model retrains on every time interval defined by the user in the presence of new data points (i.e., CPU, memory, and network usage).

Moreover, in the case that a pipeline's step requires more resources not available on its current allocated device, or if an event such as a high device load occurs, the adaptation and scheduling components initiate reallocation.

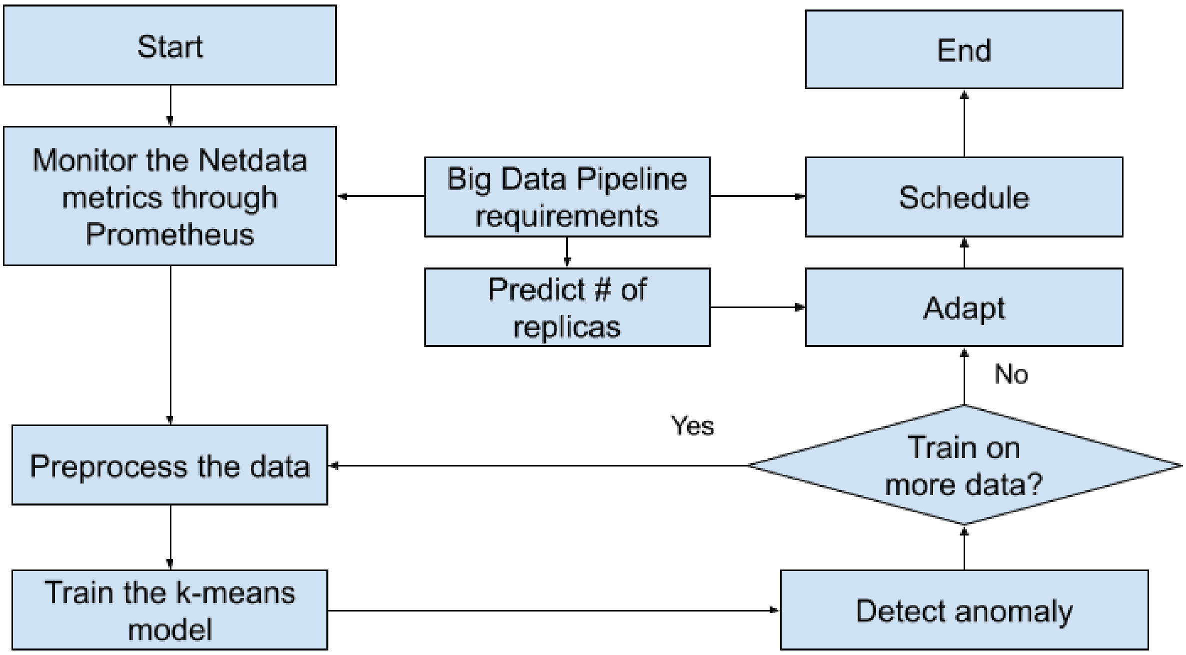


Figure 2: Flowchart of ADA-PIPE's monitoring, adaptation, and scheduling.

Co-funded by the Horizon 2020
Framework Programme of the European Union

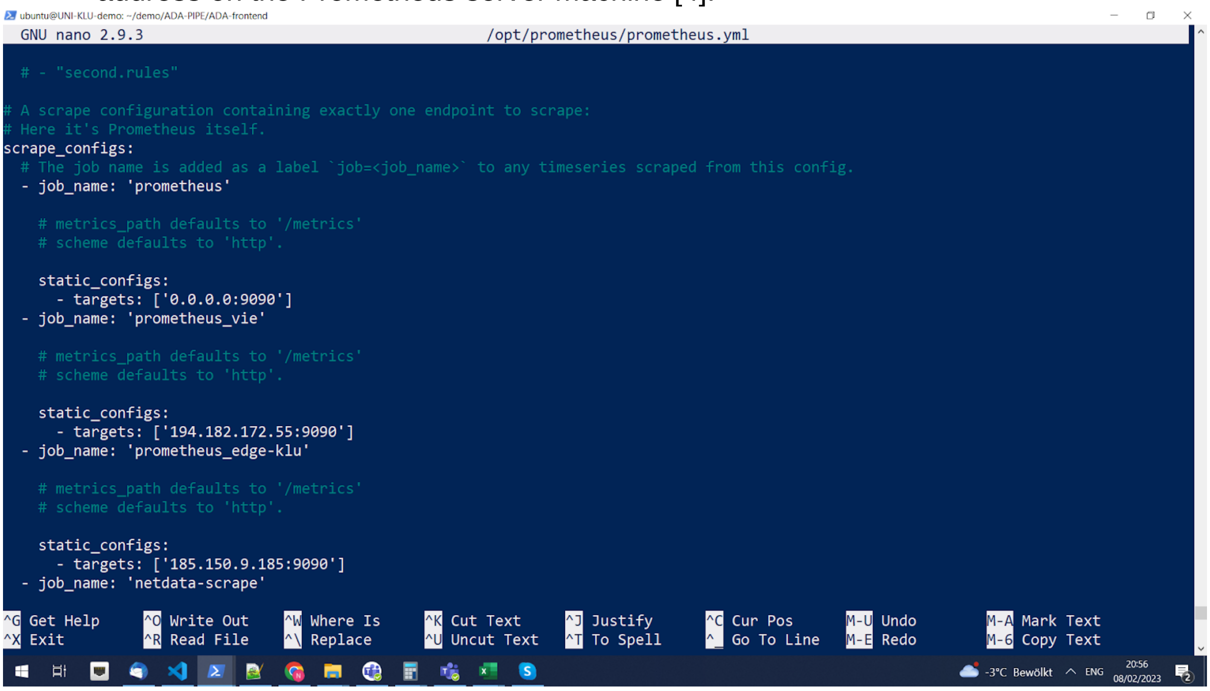
4 MONITORING

4.1 EXPORTING *NETDATA* METRICS TO *PROMETHEUS*

Prometheus, as a distributed monitoring system, offers a simple setup and a robust data model [1], [2]. We followed the tutorial in [3] to export the Netdata metrics to the Prometheus server machine and collect monitoring metrics.

Prometheus monitoring tool divides its configuration into three parts of `global`, `rule_files`, and `scrape_configs` in a file named `prometheus.yml` (see Figure 3):

- In the `global` segment, one can observe the configuration details of Prometheus. Specifically, the `scrape_interval` part governs the frequency at which Prometheus retrieves data from target devices, while the `evaluation_interval` part determines the interval at which the software assesses rules. These rules play a crucial role in generating new time series and triggering alerts.
- The `rule_files` segment denotes the location of any rule files intended for loading by Prometheus. The `rule_setting` part loads rules once and evaluates them according to the global `evaluation_interval`.
- The `scape_configs` segment provides details regarding the resources that Prometheus monitors, where we added multiple devices by configuring the target address on the Prometheus server machine [4]:



```
  GNU nano 2.9.3                    /opt/prometheus/prometheus.yml

  # - "second.rules"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ['0.0.0.0:9090']
  - job_name: 'prometheus_vie'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ['194.182.172.55:9090']
  - job_name: 'prometheus_edge-klu'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ['185.150.9.185:9090']
  - job_name: 'netdata-scrape'

^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos    M-U Undo     M-A Mark Text
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line M-E Redo     M-6 Copy Text
```

Figure 3: Scraping configuration of Prometheus monitoring system.

Figure 4: Scraping Netdata CPU utilization. shows the scraping of one Netdata metric regarding the CPU usage per core of one machine running the Prometheus:

Co-funded by the Horizon 2020
Framework Programme of the European Union

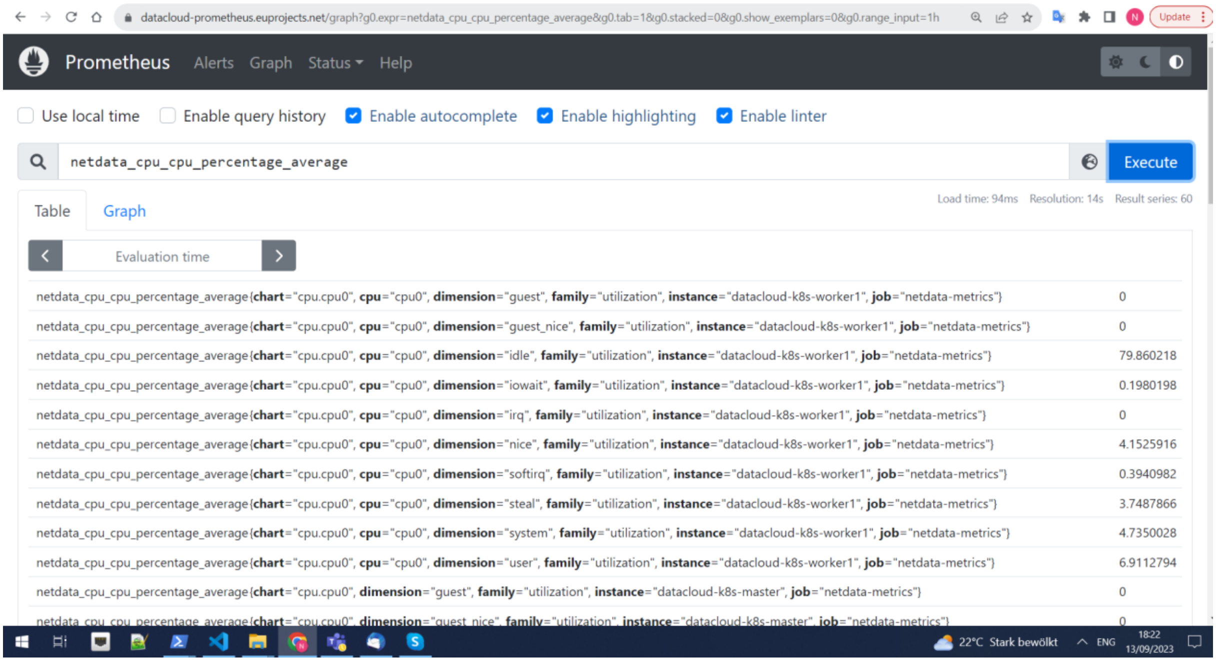


Figure 4: Scraping Netdata CPU utilization.

Moreover, Figure 5: Monitoring Netdata K8s CPU utilization per core. shows the utilization of an `8`-core machine in the cluster. ADA-PIPE collects this monitoring information of the `k8s` orchestrating the cluster of devices.

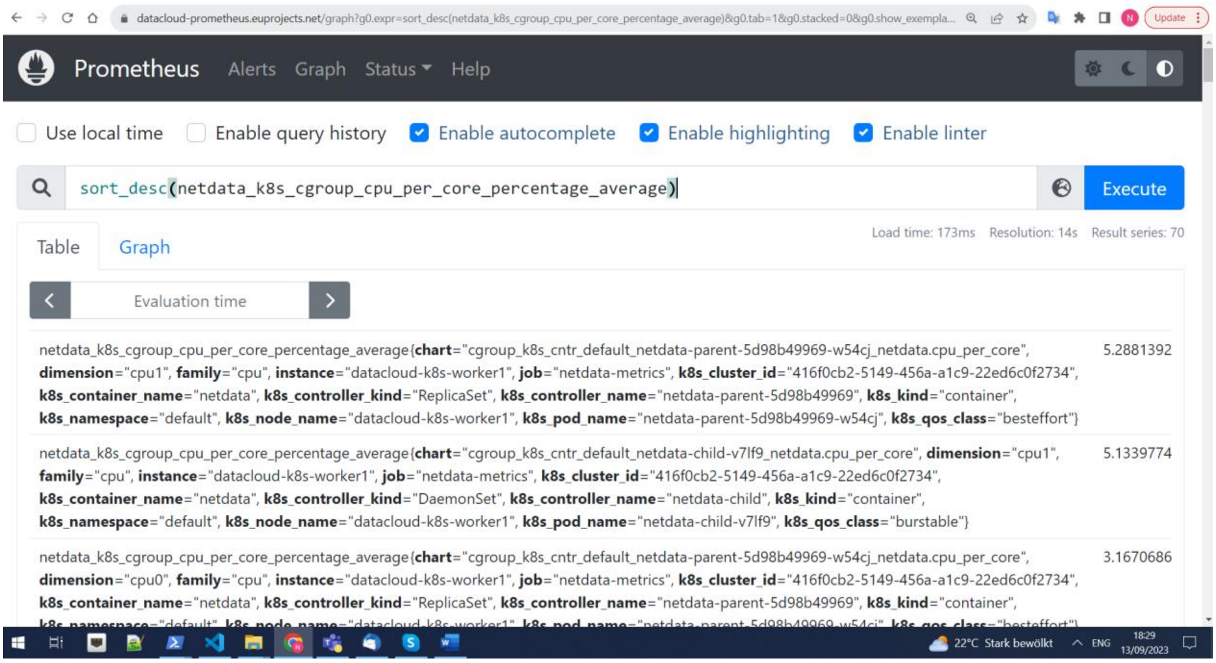


Figure 5: Monitoring Netdata K8s CPU utilization per core.

Co-funded by the Horizon 2020
Framework Programme of the European Union

`HTTP API` (i.e., `http://your.netdata.ip:19999/api/v1/`) is another method to extract the NetData metrics to send a request to the Prometheus server and receive a response with all the metrics in the JSON format:
`wget http://netdata.ip:19999/api/v1/allmetrics??format=json&filter=cpu.cpu*`
The API mentioned above requires the IP address or hostname (`your.netdata.ip`) of the Prometheus server.

4.2 POLICIES AND RULES

This section summarizes the policies and rules with respect to the resource utilization applied in Prometheus and Netdata monitoring tools.

Prometheus sends alerts based on Netdata metrics in the case of the overutilization of the processor or memory and storage overflows (see Figure 6: Prometheus policies and rules.):

- If the processor utilization is above 70%,
- If the memory utilization is above 70%,
- If the disk usage is above 80%,
- To predict if the root filesystem is going to be filled up until the next six hours.

The linear-regression-based `predict_linear(v range-vector, t scalar)` function predicts the value of the time window `t` seconds from now, based on the range vector `v`. This function can receive a time window and predict whether the filesystem will be filled up during the following six hours or not.

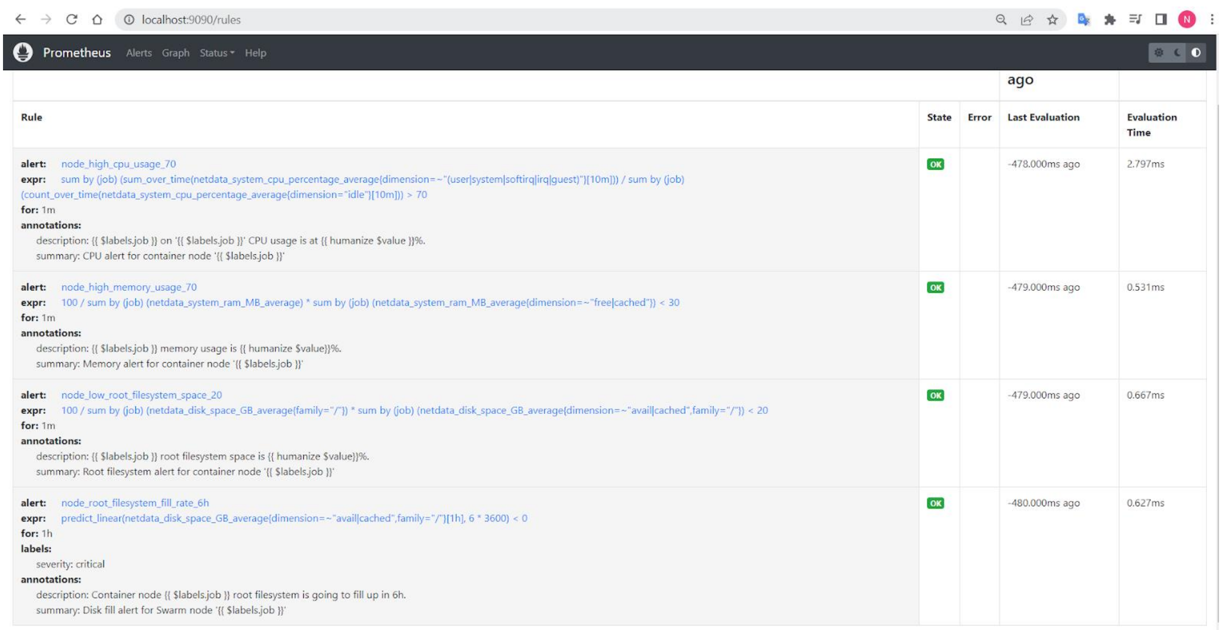


Figure 6: Prometheus policies and rules.

Figure 7: Netdata alert for warning and critical conditions. shows that Netdata first checks the utilization based on the available disk storage usage as follows:

```
$disk_utilization = $used * 100 / ($avail + $used)
```


Co-funded by the Horizon 2020 Framework Programme of the European Union

Thereafter, it checks the severity conditions by following two rules:

- Warning when: `$this > (($status >= $WARNING) ? (80) : (90))`
- Critical when: `$this > (($status == $CRITICAL) ? (90) : (98))`

where $this is equal to $disk_utilization.

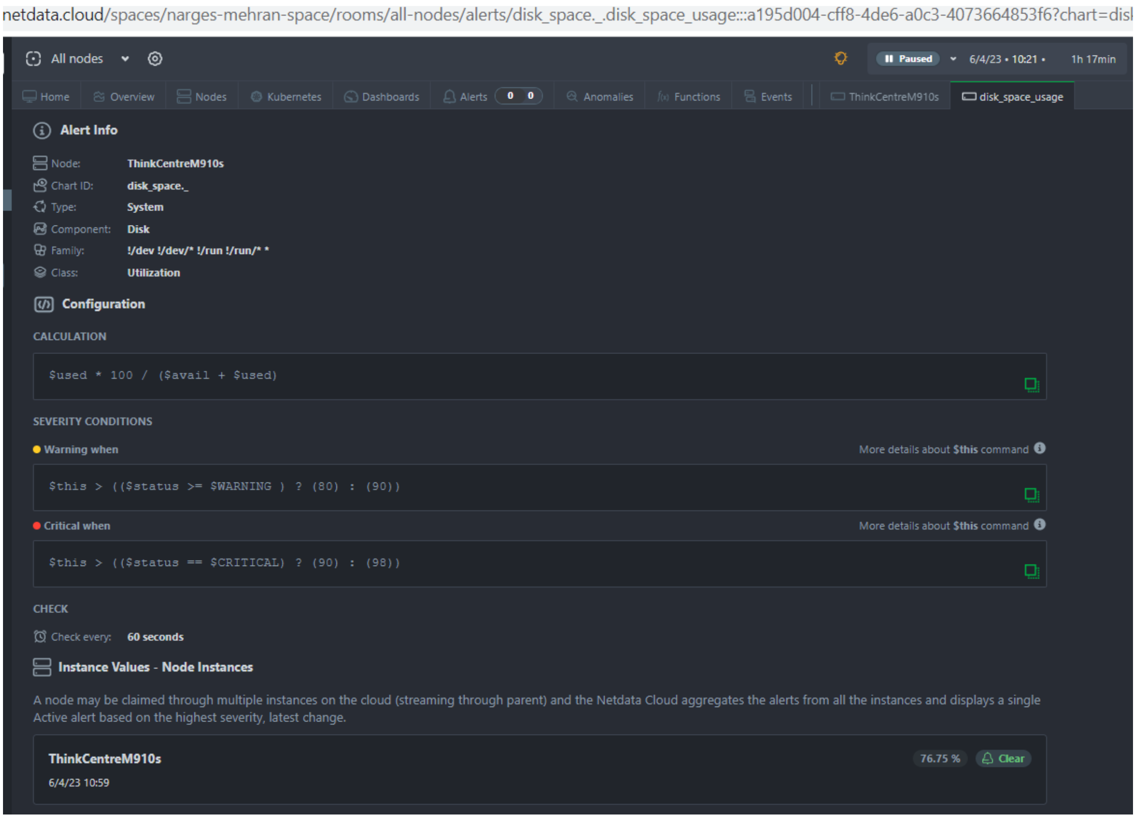


Figure 7: Netdata alert for warning and critical conditions.

Co-funded by the Horizon 2020
Framework Programme of the European Union

5 MODEL

This section presents the formal model underneath our work.

5.1 APPLICATION MODEL

1) Data pipeline workflow is a directed acyclic graph $W = (S, E, S_{src}, S_{snk})$, consisting of

   a) A set of N_S interconnected data processing steps: $S = \{s_i \mid 0 \leq i < N_S\}$.

   b) A set of data flows: $data_{ui}$ streaming from an upstage step $s_u \in S$ to a downstage step $s_i \in S$: $E = \{(s_u, s_i, data_{ui}) | (s_u,s_i) \in S{\times}S\}$. A data flow consists of a sequence of $Size_{ui}$ data elements $data_{ui}[x]$, where $1 \leq x \leq Size_{ui}$. We further denote a generic data element as e to simplify the notation.

   c) A set of producers: $S_{src} \subset S$ generating data at the rate MCRsrc that require further processing from the workflow application. A producer has no upstage steps:

      $\exists\ (s_i, src, -) \in E, \forall\ src \in S_{src} \wedge s_i \in S;$

   d) A set of consumers: $S_{snk} \subset S$ collecting and presenting the application output. A consumer has no downstage steps: $\exists\ (snk, s_i, -) \in E, \forall snk \in S_{snk} \wedge s_i \in S$.

2) Dependency level: $l(s_i)$ of a step $s_i \in S$ is the maximum number of data flows separating one step from a producer in S_{src}:

$$l(s_i) = 0,\ s_i \in S_{src};$$

$$\max(s_u,s_i,data_{ui}) \in E\ l(s_u)+1,\ s_i \in S_{src}.$$

3) Resource requirements: $req(s_i, data_{ui})$ for proper processing of data elements $data_{ui}[x]$ ($1 \leq x \leq Size_{ui}$) by a step s_i is a triple representing the minimum processing load $CPU(s_i, data_{ui})$ (in million of instructions (MI)), memory $MEM(s_i, data_{ui})$ and storage $STOR(s_i, data_{ui})$ sizes (in MB):

$$req(s_i,data_{ui})=(CPU(s_i,data_{ui}),MEM(s_i,data_{ui}),STOR(s_i,data_{ui})).$$

4) Processing time: $PT(s_i,d_j)$ or $PT_{i,j}$ of s_i on a device $d_j = sched(s_i)$ is the ratio between its computational workload $CPU(s_i)$ (in MI) and the processing speed CPU_j (in MI per second:

$$PT_{i,j} = CPU(s_i) / CPU_j$$

5) Number of replicas $R_{i,j}$ for horizontally scaling a step s_i based on the producer call rate SCR_{src} and its processing time $PT_{i,j}$ on a device d_j : $R_{i,j} = SCR_{src} \cdot PT_{i,j}$

Co-funded by the Horizon 2020 Framework Programme of the European Union

5.2 RESOURCE MODEL

We model the computing continuum as a heterogeneous set of capacity-constrained devices.

1) Devices: $D = \{d_j \mid 0 \leq j < N_D\}$ represent a set of N_D Cloud, Fog, and Edge resources in the computing continuum.

2) Capacity: $c_j = (CPU_j, MEM_j, STOR_j)$ of a device $d_j \in D$ is a vector representing its available processing speed CPU_j (in MI per second), memory MEM_j and storage $STOR_j$ sizes, depending on its utilization. A device can be in three states based on an availability threshold θ of its resources:

a) Under-utilized: indicating positive available capacity: $c_j > 0 \Longleftrightarrow CPU_j > \theta \land MEM_j > \theta \land STOR_j > \theta$;

b) Fully-utilized: indicating nearly zero free capacity: $c_j \approx 0 \Longleftrightarrow 0 \leq CPU_j \leq \theta \lor 0 \leq MEM_j \leq \theta \lor 0 \leq STOR_j \leq \theta$;

c) Over-utilized: indicating over-committed capacity: $c_j < 0 \Longleftrightarrow CPU_j < 0 \lor MEM_j < 0 \lor STOR_j < 0$.

Co-funded by the Horizon 2020
Framework Programme of the European Union

6 ADAPTATION AND SCHEDULING

6.1 DATA PREPROCESSING

A “feature vector” is used for training the ML model along with the prediction.

- ADA-PIPE first takes differences for data points that have trends in their values.
- ADA-PIPE then smooths the values a bit so that things work a bit better with metrics that can tend to be a bit spikey. Smoothing the values consists of taking a moving average (rolling average) as a calculation to analyze data points by creating a series of averages of different selections of the full dataset.
- This is the final feature vector. So Netdata anomaly detection works on a differenced and smoothed collection of recent measurements.

Algorithm 1. Data preprocessing.			
Input:	dataset: monitoring data for devices in different time_intervals		
	num_data_points_to_diff: number of data_points to take their differences		
	num_data_points_to_smooth: number of data_points to be smoothed		
	num_data_points_to_lag: number of lagged data_points		
Output:	dataset: differenced, smoothed, and lagged data_points in dataset		
function preprocessData			
	for all (vector in dataset) **do:**		
		for all (data_point in vector) **do:**	
			Take the difference of the num_data_points_to_diff subsequent data_points
			Take the rolling average (moving average) over num_data_points_to_smooth data_points
			Include the num_data_points_to_lag latest data_points in addition to the most recent one as a feature vector
		end for	
	end for		
	return dataset		
end function			

Co-funded by the Horizon 2020 Framework Programme of the European Union

6.2 k-MEANS MACHINE LEARNING ALGORITHM

Clustering a dataset involves identifying groups of similar data points, which can be achieved using clustering algorithms. Among the widely used clustering algorithms, unsupervised learning-based methods such as k-means clustering [5] operate by maintaining a set of k centroids that perform as representatives for the clusters. Thereafter, the k-means method groups data into clusters and identifies the points that are away from the clusters' centroids. A data point is assigned to a particular cluster if it is closer to the centroid of the cluster compared to other centroids. The process of finding optimal centroids in k-means involves iterative steps. It alternates between two procedures: (1) assigning data points to clusters based on the current centroids and (2) updating the centroids by selecting points that serve as the centroid of each cluster, which considers the current assignment of data points. In summary, this algorithm aims to divide the observations into k clusters in which each observation belongs to a specific cluster with the nearest mean (cluster centers or cluster centroid), serving as a prototype of the cluster. Then, it sorts the distance for each data point and determines k nearest neighbours based on minimum distance values. Afterward, it analyzes the category of those neighbours and assigns the category for the test data based on the majority vote. Finally, it returns the predicted class. The k-means clustering method is based on either Lloyd's or Elkan's algorithm. The average complexity of k-means clustering algorithm is by $O(knR)$, where n is the number of samples, and R is the number of iterations.

6.2.1 Elbow method

The Elbow method operates on the principle of conducting k-means clustering on a range of several clusters (e.g., [1-15]). For each value, we calculate the sum of squared distances from each data point to its assigned centroid, known as distortions. By plotting these distortions and inspecting the resulting curve, if we observe a bend resembling an arm, the "elbow" or point of inflection indicates the optimal value for 'k'. Figure 8 shows that we can set the number of clusters to the user-defined k=5 to fine-tune the model's parameters.

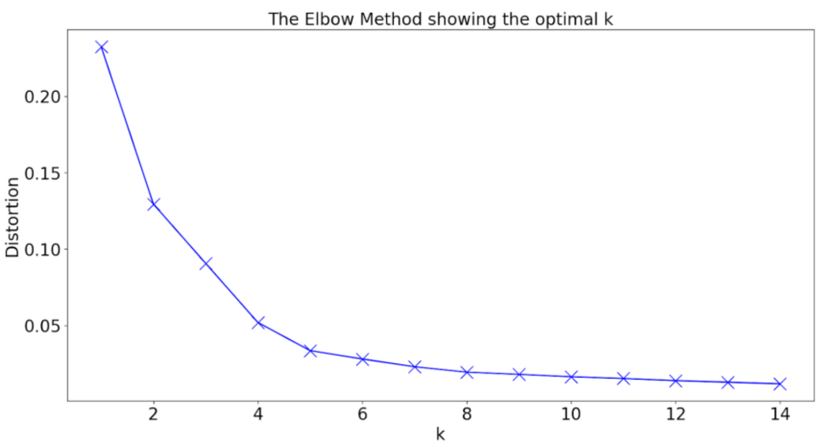


Figure 8: Elbow method to estimate the number of clusters in k-means.

Co-funded by the Horizon 2020
Framework Programme of the European Union

6.3 ANOMALY DETECTION

Anomaly refers to an atypical pattern in a dataset that deviates from the anticipated one. The anomalies can manifest within a system for various reasons, encompassing software failures, hardware malfunctions, system overloads, human errors, and noise [7]. The objective is to transform the computed raw anomaly score into a value that lies within the range `[0-1]`. In this context, a normalized score of `1` equates to a magnitude comparable to the most significant observed distance, which signifies the utmost anomaly within the training data. In other words, scores exceeding 99% reflect anomalies that are equally or even more pronounced than the most unusual 1% instances observed during training. Therefore, ADA-PIPE utilizes the minimum and maximum raw scores observed during the training phase.

Henceforth, ADA-PIPE converts this raw distance measure into an anomaly score by "normalizing" based on the maximum distance observed during training. For example, if the max distance during training was 120 and the min was 80, and the distance for the most recent vector was 125, then the normalized anomaly score would be (125 - 80) / (120 - 80) ~= 113%. By default, any score larger than 99% is anomalous.

To try and make use of our zero-and-one matrix, we need to determine the number of ones in a recent rolling window on our device. If the anomaly rate among all metrics at a specific time interval is above a threshold, we flag the device itself as anomalous. In the work, we use a 10% anomaly rate threshold. The anomaly detector maintains a rolling window of "Device Anomaly" values. Once the device is detected as anomalous for a period within this rolling window, ADA-PIPE triggers an anomaly event while the device anomaly counter stays above the threshold.

6.3.1 k-means-based anomaly detection

1. Determine the value of `k` using the Elbow method and specify an optimal number of clusters, denoted as `num_of_clusters`.
2. Randomly assign `num_of_clusters` centroids.
3. Compute the coordinates of the cluster centroids.
4. Calculate the distances between each data point and the centroids and assign each point to the nearest cluster centroid based on the minimum distance.
5. Recalculate the cluster centroids.
6. Repeat steps 4 and 5 iteratively until the model reaches the global optima, where no further improvement is possible, and no data points switch from one cluster to another.
7. Loop over each row of data and produce anomaly scores once we have a base-trained model to retrain periodically as defined by 'train_interval' [6].

Co-funded by the Horizon 2020
Framework Programme of the European Union

Algorithm 2. k-means-based anomaly detection.	
Input:	dataset: monitoring data for devices in different time_intervals
	train_interval
	num_data_points_to_train
	num_data_points_to_diff: number of data_points to take their differences
	num_data_points_to_smooth: number of data_points to be smoothed
	num_data_points_to_lag: number of samples to lag
Output:	anomaly_score
	anomaly_bit
function detectAnomalies	
	dataset ← preprocessData(dataset, num_data_points_to_diff, num_data_points_to_smooth, num_data_points_to_lag)
	num_of_clusters ← Elbow_method(dataset)
	for all (vector of data_points in a dataset) **do:**
	for all (data_point in vector) **do:** ▷ ***Train / Re-Train***
	if (time_interval >= num_data_points_to_train) & (time_interval % train_interval == 0) **then:**
	Randomly assign num_of_clusters centroids
	Compute the coordinates of the cluster centroids
	while (changing in the centroids of clusters) **do:**
	for all (data_point in the clusters) **do:**
	Calculate the distances between each data point and the centroids
	Assign each point to the nearest cluster centroid based on the minimum distance
	end for
	for all (cluster) **do:** ▷ **calculate the center of gravity for each cluster**
	Recalculate the cluster centroids
	Calculate the min and max anomaly scores
	end for

Co-funded by the Horizon 2020 Framework Programme of the European Union

				end while
			else: ▷ **Inference / Scoring**	
			▷ ***If we have a fitted model, then predict anomaly score***	
				Get the existing trained cluster centroids
				Get anomaly score based on the sum of the Euclidean distances between the feature vector and each cluster centroid
				Score the data point anomaly-based on min-max normalization and min and max anomalies
				Calculate anomaly bit based on anomaly score
			end if	
		end for		
	end for			
	return (anomaly_score, anomaly_bit)			
end function				

6.4 ADAPTATION

This component defines the adaptation based on the anomalous or normal scores of the computing devices and requirements of the pipeline's steps.

Algorithm 3. Updating the computing continuum.		
	Input:	dataset: monitoring data for devices in different time_intervals
		train_interval
		num_data_points_to_train
		num_data_points_to_diff: number of data_points to take their differences
		num_data_points_to_smooth: number of data_points to be smoothed
		num_data_points_to_lag: number of samples to lag
		D: Device set
	Output:	D: updated device set
		Anomal: Anomalous device set
	function updateContinuum	

Co-funded by the Horizon 2020
Framework Programme of the European Union

	Anomal ← ∅
	anomaly_score, anomaly_bit ← detectAnomalies(dataset, train_interval, num_data_points_to_diff, num_data_points_to_smooth, num_data_points_to_lag)
	for all $d_j \in D$ **do:**
	if anomaly_bit[j]=1 **then:**
	$D \leftarrow D \setminus d_j$ ▷ ***Remove from the list of devices***
	Anomal ← Anomal U $\{d_j\}$ ▷ ***Add it to the list of anomalous devices***
	return (D, Anomal)
end function	

Figure 2 illustrates the ADA-PIPE's component with its replica prediction, consisting of seven components. a) Microservice requirement analysis: receives resource requirements req (mi) of the microservices of user's application to update the trace;

b) Microservice trace: consists of rows with the timestamp, microservice name, microservice container instance identifier, and the collected metrics (i.e., SCR_{src}, $PT_{i,j}$) [3];

c) Feature set: receives the dataset and extracts the microservice call rate SCR_{src} for a corresponding microservice time $ST_{i,j}$. The ML models learn to fit the MT as the input to the SCR_{src} as the output;

d) ML hyperparameter design: component receives the feature set consisting of SCR_{src} and $ST_{i,j}$ for fine-tuning and optimizing the ML models;

e) Prediction model: forecasts the step call rate based on the processing time by utilizing the ML prediction model of gradient boosting regression;

f) Replica: component estimates the required instances to scale out from each step based on the multiplication between its predicted call rate SCR'_{src} and time $ST_{i,j}$;

g) Orchestration: manages the microservices on the Cloud virtual machines by utilizing the Kubernetes replica scaling based on the predicted microservice call rates and decisions taken by the integrated scheduler [REF].

Algorithm 4. Predicting the number of replicas.	
Input:	Processing times of pipeline steps on the devices
	Number of calls for each pipeline's step
Output:	Number of replicas
function predictRepilicas	

Co-funded by the Horizon 2020 Framework Programme of the European Union

2		**do:** ▷ ***Validate the ML model***	
3			Tune hyperparameters of ML model
4		**while** (ML model converges to lowest loss)**;**	
5		Fit an ML model to the dataset	
6		Predict the pipeline's step calls	
7		Calculate the number of replicas for each pipeline's step	
		return number of replicas	
8	**end function**		

6.5 SCHEDULING

C3-Match depicted in Algorithm 5 applies matching game principles to schedule the steps of a data pipeline A on the continuum devices D. In addition, C3-Match receives availability threshold vector Θ of resources. Firstly, the algorithm calculates an array of dependency levels L in lines 5–8, where each element L[l] represents a set of independent steps separated from a producer by a maximum of l edges. Then, the algorithm iterates over the dependency levels to create the device preference lists for its independent steps (line 10) and the step preference lists for the devices (line 11). Line 12 finds appropriate mappings for each step to the preferred devices and allocates the required memory and storage based on its available capacity.

Algorithm 5. C3-Match scheduling.			
	Input:	W: Data pipeline	
		D: Set of devices	
		L: Step set of a dependency level	
	Output:	DPL [s_i], $\forall s_i \in$ L: Device preference lists of steps	
1	**function schedule**		
2		W ← predictRepilicas	
3		D ← updateContinuum	
4		sched(s_i) ← ∅, $\forall s_i \in$ L ▷ ***Initialize schedule***	
5		**for all** ($s_i \in$ L) **do** ▷ ***Calculate dependency levels***	
6			$l(s_i) \leftarrow \max(s_u,s_i,data_{ui})\in E\ l(s_u) + 1$

Co-funded by the Horizon 2020
Framework Programme of the European Union

Algorithm 5. C3-Match scheduling.		
	Input:	W: Data pipeline
		D: Set of devices
7		$L[l(s_i)] \leftarrow L[l(s_i)] \cup \{s_i\}$ $\triangleright$ ***Add*** s_i ***to its corresponding level***
8		**end for**
9		**for all** ($l \in [1, \text{sizeof}(L)]$) **do:** $\triangleright$ ***Iterate steps***
10		DPL ← rankDevices(A, D, L[l], Θ) $\triangleright$ ***Call Algorithm 6***
11		SPL ← rankSteps(A, D, L[l], DPL) $\triangleright$ ***Call Algorithm 7***
12		sched ← match(A, D, L[l], DPL, SPL, sched, Θ) $\triangleright$ ***Call Algorithm 8***
13		**end for**
14		**return** sched
15	**end function**	

6.5.1 Step-side ranking

This component orders the devices for the pipeline's steps regarding the resource utilization of devices. Step-side ranking, presented in Algorithm 6, receives the data pipeline application A, a set of devices D, a step set L of a certain dependency level, and availability threshold vector Θ of resources. The algorithm initializes the empty device preference lists $DPL[s_i]$ for every step s_i in line 1. Afterward, each step ranks the devices (lines 2–9) by first filtering those that do not have sufficient resources for the step s_i (line 5). Then, it creates a list of tuples for each step s_i that associates a device d_j with the maximum requirements (line 6). Finally, line 11 sorts the device preference lists of each step in descending order based on its requirement ***req***$(s_i, data_{ui})$.

Algorithm 6. Step-side ranking.		
	Input:	W: Data pipeline
		D: Set of devices
		L: Set in a dependency level
	Output:	$DPL[s_i], \forall s_i \in L$ $\triangleright$ ***Device preference lists of steps***
1	**function rankDevices**	
2		$DPL[s_i] \leftarrow \emptyset, \forall s_i \in L$ $\triangleright$ ***Initialize DPL***
3		**for all** ($s_i \in L$) **do:** $\triangleright$ ***Iterate steps***

Co-funded by the Horizon 2020 Framework Programme of the European Union

4		**for all** ($d_j \in D$) **do:**
5		**if** (CORE (s_i) < CORE$_j$) ∧ (MEM (s_i) < MEM$_j$) ∧ (STOR (s_i) < STOR$_j$) **then** ▷ ***Check resource availability***
6		DPL[s_i] ← DPL[s_i] U (d_j, max $\forall(s_u,s_i,data_{ui}) \in E$ req(s_i, $data_{ui}$))
7		**end if**
8		**end for**
9		**end for**
10		**for all** $s_i \in L \wedge$ DPL[s_i] = ∅ **do**
11		DPL[s_i] ← Sort***req*** (DPL[s_i]) ▷ ***Rank based on req(***s_i***)***
12		**end for**
13		**return** DPL
14	**end function**	

6.5.2 Device-side ranking

This component orders the pipeline's steps for the devices in regard to the steps' resource requirements. The device-side ranking, presented in Algorithm 7, receives as input the device preference lists DPL[s_i] ($\forall s_i \in S$) computed in Algorithm 6, along with the data pipeline W, the device set D, and steps L of a specific dependency level. Similarly, the algorithm initializes the empty step preference lists SPL[d_j] for each device d_j in line 2. Afterward, each device in the preference list DPL[s_i] of each step s_i creates, in lines 3–9, a step preference list SPL[d_j] associating to each step s_i the maximum requirements among all its upstage steps s_u (line 5). Finally, line 9 sorts the step preference lists in descending order based on the requirements ***req***(s_i, $data_{ui}$).

Algorithm 7. Device-side ranking.		
	Input:	W: Data pipeline
		D: Set of devices
		L: Set in a dependency level
		DPL[s_i], $\forall s_i \in L$ ▷ ***Device preference lists of steps in L***
	Output:	SPL[d_j], $\forall d_j \in D$ ▷ ***Step preference lists of devices***
1	**function rankSteps**	
2		SPL [d_j] ← ∅, $\forall d_j \in D$ ▷ ***Initialize DPL***
3		**for all** ($s_i \in L$) **do:** ▷ ***Iterate steps***

Co-funded by the Horizon 2020 Framework Programme of the European Union

6		**for all** ($d_j \in D$) **do:**
8		$SPL[d_j] \leftarrow SPL[d_j] \cup (s_i\ ,\ \max_{(\forall(s_{u,si,datau i})\in E)} cap(s_i, data_{ui}))$
9		**end for**
10		**end for**
11		**for all** $s_i \in L \wedge SPL[d_j] = \emptyset$ **do**
12		$SPL[d_j] \leftarrow$ Sort***Cap***($SPL[d_j]$) ▷ ***Rank based on cap***
13		**end for**
14		**return** SPL
15	**end function**	

6.5.3 Matching-based scheduling

Capacity-aware matching, presented in Algorithm 8, matches the steps of a dependency level to the devices based on their mutual preference lists computed in Algorithms 6 and 7. The goal is to identify a schedule that maximizes the aggregate step-side utility of the data pipeline and device-side utility of the resource provider. After initializing an empty step allocation list for each device (line 1), the algorithm iterates over the steps set L in a dependency level (line 2), identifies the highest ranked device d_j for each step, and retrieves its resources (lines 3-4). Then, if the device d_j has also ranked the step in its preference list $SPL[d_j]$ (line 6), the algorithm continues in one of the following three states based on the capacity c_j of a device d_j (line 5).

Algorithm 8. Capacity-aware matching.		
	Input:	W: Data pipeline
		D: Set of devices
		L: Set in a dependency level
		$DPL[s_i], \forall s_i \in L$ ▷ ***Device preference lists of steps in L***
		$SPL[d_j], \forall d_j \in D$ ▷ ***Step preference lists of devices***
		$sched(s_i), \forall s_i \in L$ ▷ ***Schedules***
	Output:	$sched(s_i), \forall s_i \in L$ ▷ ***Updated schedules***
1	**function match**	
2		$alloc[d_j] \leftarrow \emptyset, \forall d_j \in D$
3		**for all** ($s_i \in L$) $\wedge$ ($L = \emptyset$) **do** ▷ ***Allocate all steps in a level***
4		$d_j \leftarrow FIRST(DPL[s_i])$

Co-funded by the Horizon 2020 Framework Programme of the European Union

5	$c_j \leftarrow (CPU_j, MEM_j, STOR_j)$ ▷ ***Resources of device d_j***
6	**if** $s_i \in SPL[d_j]$ **then** ▷ ***Check if step is also ranked by device***
7	**if** $c_j > \Theta$ **then** ▷ ***State a): Under-utilization***
8	sched $(s_i) \leftarrow d_j$ ▷ ***Schedule s_i on d_j***
9	$alloc[d_j] \leftarrow$ Sort***cap*** $(alloc[d_j] \cup s_i)$ ▷ ***Add s_i to allocation list***
10	$c_j \leftarrow c_j - req\ (s_i, data_{ui})$ ▷ ***Allocate device capacity***
11	**end if**
12	**if** $c_j < \Theta$ **then** ▷ ***State b): Over-utilization***
13	$s_l \leftarrow Last(alloc[d_j])$ ▷ ***Select s_l with requirement***
14	$sched(s_l) \leftarrow \emptyset$ ▷ ***Unschedule s_l from d_j***
15	$alloc[d_j] \leftarrow alloc[d_j] \setminus s_l$ ▷ ***De-allocate s_l***
16	$c_j \leftarrow c_j + req\ (s_l, data_{l'l})$ ▷ ***Free device capacity***
17	$L \leftarrow L \cup s_l$ ▷ ***Return s_l to unscheduled step set L***
18	**end if**
19	**if** $c_j \approx \Theta$ **then** ▷ ***State c): Full-utilization***
20	$s_l \leftarrow Last(alloc[d_j])$ ▷ ***Select s_l with highest transmission***
21	**for all** $s_s \in SPL[d_j] \wedge SPL[d_j].Index(m_l) < SPL[d_j].Index(s_s)$ **do**
22	$SPL[d_j] \leftarrow SPL[d_j] \setminus s_s$ ▷ ***Remove ss with higher transmission***
23	$DPL[s_s] \leftarrow DPL[s_s] \setminus d_j$ ▷ ***and corresponding device d_j***
24	**end for**
25	**end if**
26	**end if**
27	**end if**
28	**end for**
29	**return** sched
30	**end function**

Underutilization (lines 6–10) first checks the resource capacity of a device. If the device is under-utilized, it temporarily matches the step to the preferred device (line 8) and sorts the allocation list of the device based on the maximum resource requirements of each step. After

Co-funded by the Horizon 2020 Framework Programme of the European Union

allocating the required resources (line 9), the algorithm updates the capacity c_j of the device (line 10).

Overutilization (lines 12–17) occurs if a matched step s_i exceeds the capacity c_j of device d_j (line 12). In this case, the algorithm releases resources for the step s_i by selecting the step s_l with the highest requirement in the allocation list alloc[d_j] of the device d_j (line 13). Afterward, d_j rejects its existing match the step s_l, removes it from the allocation list (lines 14–15), increases the capacity c_j (line 16), and returns it to the step set.

Full utilization (lines 19–25) occurs if a device d_j reaches its capacity c_j. The algorithm identifies the step s_l with the highest in the allocation list alloc[d_j] of the device d_j (line 20). Afterward, d_j removes the steps ss with a higher requirement than the temporarily-matched step s_l (line 21) from its preference list SPL[d_j] (line 22). In this case, the step ss has a lower rank with a larger index than s_l in the preference list SPL[d_j]. Similarly, the step ss removes d_j from its device preference list DPL[s_s] (line 23), which avoids scheduling the steps with high execution performance on d_j. Furthermore, it allows scheduling higher-ranked steps in SPL[d_j] on the device d_j, gradually approaching a stable matching (schedule) by fixing the temporary matches (lines 21–24).

Co-funded by the Horizon 2020
Framework Programme of the European Union

7 EXPERIMENTAL DESIGN

7.1 NETDATA DESIGN

We used the `Python`-based `netdata-pandas` library to collect and process data. The library supports pulling data from `netdata api` into a `pandas dataframe`:

- `sudo snap install jupyter`
- `sudo apt install jupyter-core`
- `pip3 install --upgrade --force-reinstall --no-cache-dir jupyter`
- `pip3 install -U scikit-learn scipy matplotlib seaborn netdata-pandas`
- `wget https://raw.githubusercontent.com/netdata/netdata/master/ml/notebooks/netdata_anomaly_detection_deepdive.ipynb`
- `jupyter notebook --no-browser --port=8888`

Then, we specifically set one host to be monitored: 194.182.187.139:19999, and analyzed the output through the jupyter notebook project [6][8]. It is possible to extract the required Netdata metrics into a `pandas dataframe`. Thereafter, ADA-PIPE stores the data of monitored resources in `comma-separated values (CSV)` files.

7.2 DATA PREPROCESSING

The manipulation of the data is conducted in `Python 3.10` using the data analysis module `Pandas` in a two-dimensional data structure of `dataframe`. The monitoring data is sorted based on the start date of the task from earliest to latest.

We used the `diff`, `rolling`, `reindex`, and `concat` from the `pandas.dataframes` [9], [10], [11].

7.3 ANOMALY DETECTION

ADA-PIPE calculates the anomaly score based on the sum of the distances between the feature vector and each cluster centroid. The existing metrics utilized for distance calculation encompass Euclidean, Manhattan, and Minkowski measures:

Co-funded by the Horizon 2020 Framework Programme of the European Union

```
raw_anomaly_score = np.sum(cdist(X, cluster_centroids, metric='euclidean'), axis=1)
```

Therefore, we utilized the `cdist` library from the `scipy.spatial.distance` package.

Table 1: Model parameters.

Parameter Name	Parameter Value
num_of_days	30
last_n_hours	num_of_days*24
train_interval	3600
num_data_points_to_train	3600
num_data_points_to_diff	1
num_data_points_to_smooth	3
num_data_points_to_lag	5
dimension_anomaly_score_threshold	0.99
n_clusters_per_dimension	2
max_iterations	1000

charts	`system.cpu`
	`system.ram`
	`net.enp0s31f6`

dims	`system.cpu|user`
	`system.ram|used`
	`net.enp0s31f6|received`

7.4 MACHINE LEARNING PARAMETER DESIGN

We used the `kmeans` model from `scikit-learn` library to cluster the anomalies during the resource and network monitoring in a specific time interval [12]. Table 1 denotes how we set the data along with the ML parameters. These parameters are of importance in understanding how monitoring, collecting, and analyzing the model and strategy all work.

Feature preprocessing-related parameters

- num_data_points_to_diff: This is a 1 or 0 flag to turn on or off, differencing in the feature preprocessing. It defaults to 1 (to take differences) and generally should be left alone.

- num_data_points_to_smooth: The extent of smoothing (averaging) applied in feature preprocessing.

- num_data_points_to_lag: The number of previous values to include in our feature vector.

Anomaly score-related parameters

- feature vector: A feature vector is what the ML model is trained on and uses for prediction. The simplest feature vector would be just the latest raw dimension value itself [Y]. By default, Netdata will use a feature vector consisting of the six latest differences and smoothed values

Co-funded by the Horizon 2020 Framework Programme of the European Union

of the dimension, so conceptually, something like [avg3(diff1(Y-5)), avg3(diff1(Y-4)), avg3(diff1(Y-3)), avg3(diff1(Y-2)), avg3(diff1(Y-1)), avg3(diff1(Y))] which ends up being just six floating point numbers that try and represent the "shape" of recent data.

- anomaly_score: At prediction time, the anomaly score is just the distance of the most recent feature vector to the trained cluster centroids of the model, which are feature vectors, albeit supposedly the best most representative feature vectors that could be "learned" from the training data. So, if the most recent feature vector is very far away in terms of Euclidean distance, it's more likely that the recent data it represents consists of some strange pattern not commonly found in the training data.

- dimension_anomaly_score_threshold: The threshold on the anomaly score, above which the data of the dimension is considered anomalous, and the anomaly bit is set to 1 (it is actually set to 100, but this just acts more as a rate when aggregated in the Netdata agent API). By default, this is 0.99, which means anything with an anomaly score above 99% is considered anomalous. Decreasing this threshold makes the model more sensitive and will leave more anomaly bits; increasing it does the opposite.

- anomaly_bit: If the anomaly score is greater than a specified threshold, then the most recent feature vector, and hence the most recent raw data, is considered anomalous. Since storing the raw anomaly score would essentially double the amount of storage, we just store the anomaly bit in the existing internal Netdata representation without any additional overhead.

- dimension_anomaly_rate: The anomaly rate of a specific dimension (metric) in a time interval.

- node_anomaly_rate: The anomaly rate across all dimensions/metrics of a node.

Training size parameters

- train_interval: How often to train or retrain each model.

- num_data_points_to_train: How much of the recent data to train on; for example, 3600 denotes the training on the last 1 hour of raw data. The default in the `netdata` agent currently is 14400 (last 4 hours). However, we set it to the last month of the runtime and the pipeline's execution.

Model parameters

- num_of_clusters: This is the number of clusters to fit for each model; by default, it is set to 2 such that two cluster centroids will fit each model.

Co-funded by the Horizon 2020
Framework Programme of the European Union

8 RESULTS

8.1 EVALUATION ANALYSIS

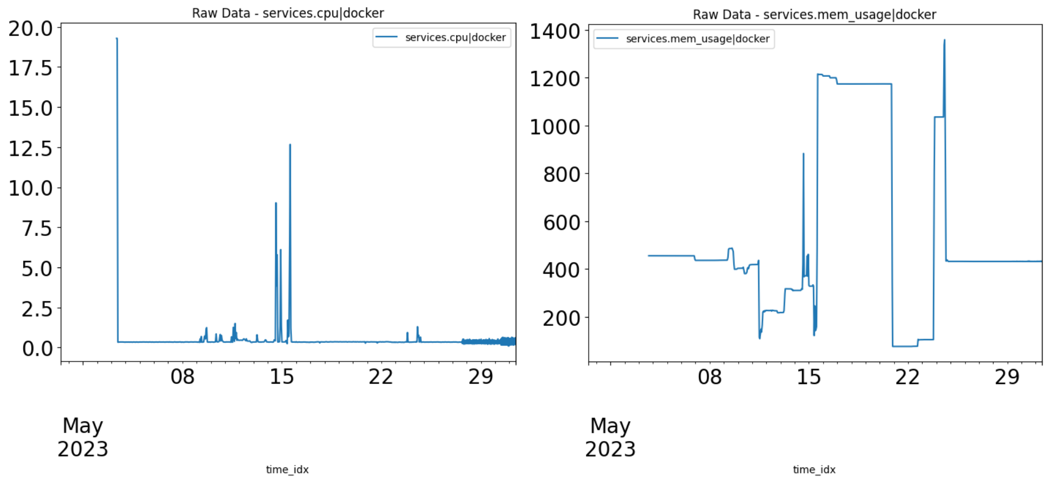
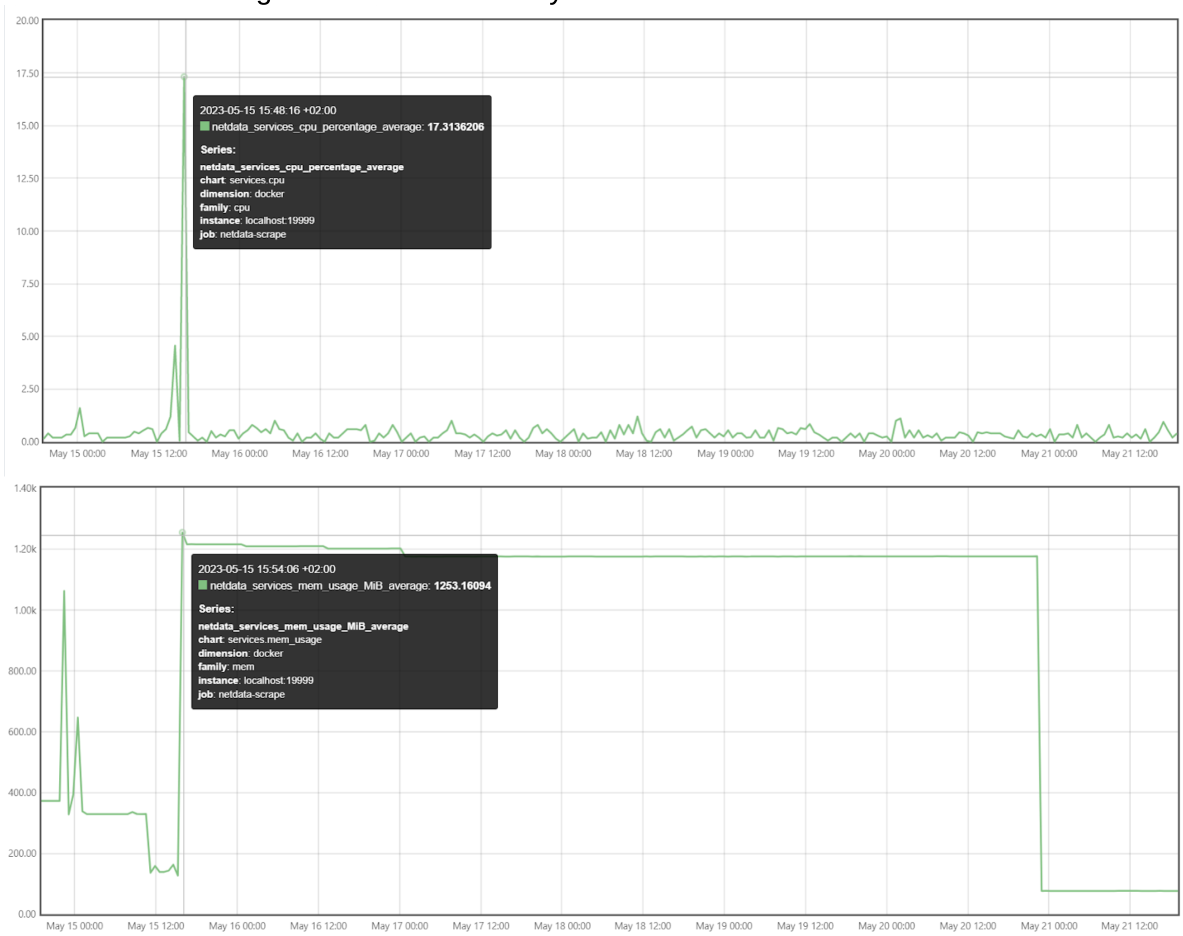
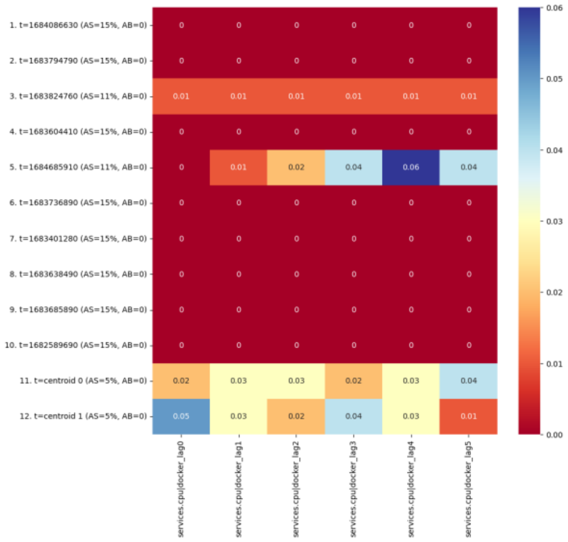


Figure. CPU and memory utilization of the Docker service.



Co-funded by the Horizon 2020
Framework Programme of the European Union

The following figures show that since the Docker service consumes a lot of memory, it can be an anomaly service.



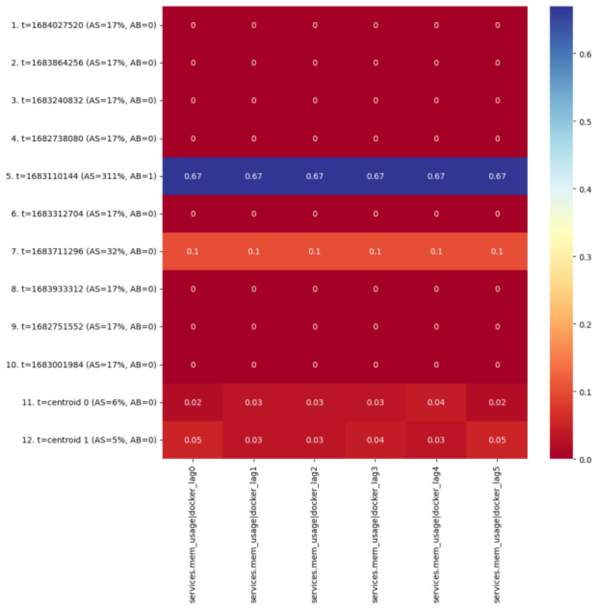


Figure. Docker service CPU utilization.

Figure. Docker service memory utilization.

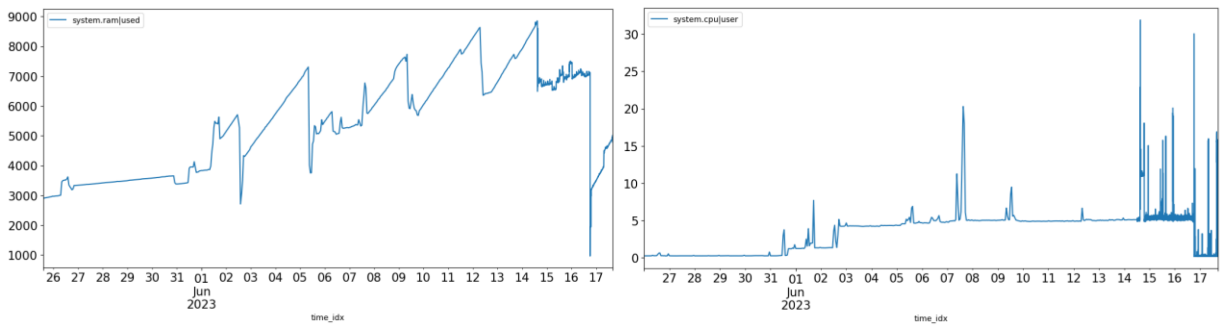
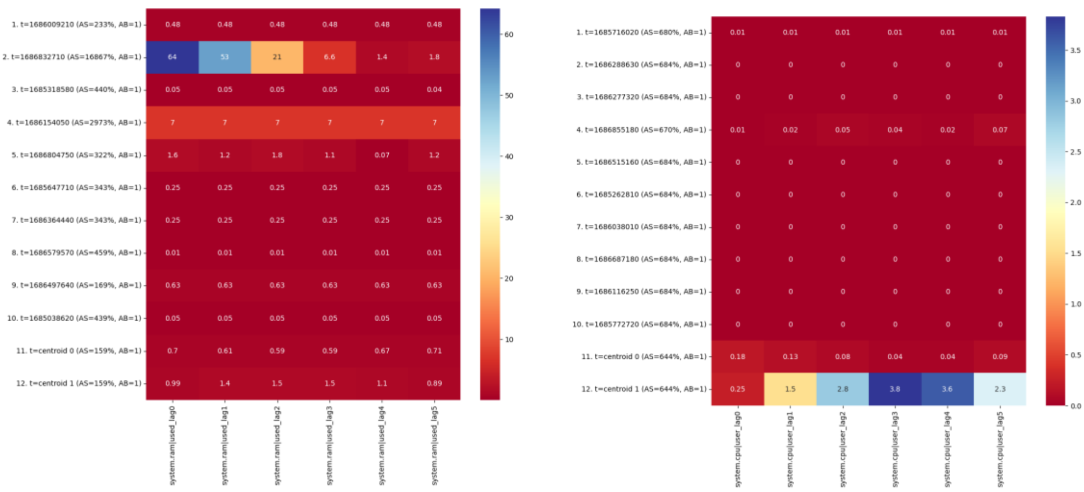


Figure. Raw data monitored of CPU and memory utilization of the device.



.

Figure. The heatmap of CPU utilization and Memory usage of the device.

Co-funded by the Horizon 2020
Framework Programme of the European Union

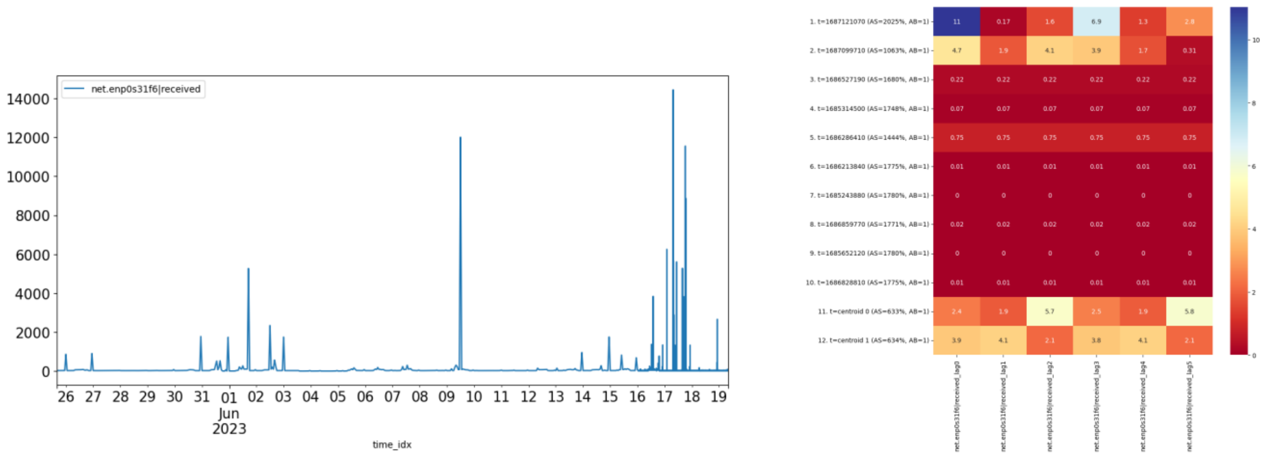


Figure. Average kilobits of data received per second by the device.

Co-funded by the Horizon 2020
Framework Programme of the European Union

9 STATE OF THE ART

Cloud-native in-production monitoring tools. Prometheus, as an open-sourced monitoring system, extracts time series data from cloud-native applications and collects metrics via the PromQL query language. Netdata enables users to quickly identify and troubleshoot issues and make data-driven decisions according to the pre-built visible dashboards. Stackdriver is Google's logging and monitoring method tightly integrated into Google Cloud. cAdvisor [13], short for Container Advisor, is an open-source tool to monitor containers developed by Google. It can collect, aggregate, process, and export container-based metrics such as CPU and memory usage, filesystem, and network statistics. Prometheus has also provided support for using cAdvisor [14], in which the user needs to configure Prometheus to scrape metrics from cAdvisor.

Customized monitoring tools. The works targeting the design of monitoring systems focus on data analytics, tracing bugs, and visualization, while monitors are tightly coupled with the monitored applications and devices. Distinct from these works, ZERO [15] designed a monitoring method decoupled from the monitored devices to minimize overhead. López-Peña et al. [16] proposed a method for the maintenance of IoT systems from operations to development.

Autoscaling. Kubernetes autoscaler [17] proposes two strategies of horizontal and vertical scaling. The horizontal pod autoscaler decides according to the current number of replicas for each service, along with the current and desired metric values to scale in/out the number of pipeline steps. However, vertical pod autoscaler considers scaling up/down the services by using statistics over a moving time interval (e.g., in the case that the memory utilization reaches the 99th percentile over the time interval of 24 hours). Autopilot [18] proposes a method to scale in/out the number of replicas from each step over an averaging window based on the CPU usage (the default is 5 minutes) and the required average utilization of the services. Rossi et al. [19] proposed a reinforcement learning-based method that takes the scaling decision based on the step's response time. Toka et al. [20] presented a proactive scaling method, including multiple ML-based forecast models to optimize resource over-provisioning and service level agreement.

Anomaly detection. Considering the growing number of data sources, anomaly detection with hierarchical temporal memory as the online, unsupervised method [21] gained a lot of attention. Moreover, the Numenta anomaly benchmark (NAB) is an open-source environment specifically designed to evaluate anomaly detection algorithms for real-world use. Lavin and Ahmad [21][22] proposed the Numenta Anomaly Benchmark (NAB), which attempts to provide a controlled and repeatable environment of open-source tools to test and measure anomaly detection algorithms on streaming data. This method aims to test and score the efficacy of real-time anomaly detectors adequately. Zhang et al. [23] proposed a Mann-Kendall-based method that models entropy-based feature selection of transformed metrics. This method aims to improve the efficiency of model training and anomaly detection, along with reducing false positives in the detection phase. Moreover, there exist several tools, such as application response measurement (ARM) and Newrelic APM, for monitoring service and device performance. The application programming interface (API) addresses this requirement by enabling the measurement of resource utilization, workload, and service

Co-funded by the Horizon 2020
Framework Programme of the European Union

measurements [24]. Newrelic APM cloud-based tool [25] allows websites and mobile apps to track user interactions and service and device performance.

Moreover, in statistical analysis, a widely utilized measure for detecting outliers in a normal distribution is known as the z-score [26]. This method involves leveraging historical data over a specific timeframe to compute a z-score for new data points. The calculation of the z-score is performed using the following formula:

$$\text{z-score} = (\text{current\_value} - \text{avg\_over\_time}) / \text{std\_dev\_over\_time}$$

Applying this formula to detect anomalies, z-scores that are in the range of [-1,1], [-2,2], and [-3,3] map as healthy, degraded, and critical states of the device, respectively.

Based on the monitoring of server metrics (e.g., CPU, memory, disk I/O, and network latency), represented by a time-series, Agrawal et al. [27] present a self-adaptive anomaly detection method designed to identify unusual behaviors. The proposed method involves analyzing log files and calculating reconstruction errors to dynamically adjust the threshold value, thereby enhancing the accuracy of anomaly detection. The method comprises five distinct steps: (1) preprocessing, (2) metric collection, (3) feature extraction, (4) prediction, and (5) anomaly detection. The authors evaluated their models' ability to predict anomaly precisely by five metrics: precision, recall, false positive rate, true positive rate, and F-measure.

Co-funded by the Horizon 2020
Framework Programme of the European Union

10 CONCLUSIONS

This deliverable presents the documentation regarding the latest version of the ADA-PIPE tool. It provides the architecture along with the ...

Co-funded by the Horizon 2020
Framework Programme of the European Union

REFERENCES

[1] Turnbull J. Monitoring with Prometheus. Turnbull Press. Jun 12, 2018. [link]

[2] https://prometheus.io/docs/prometheus/latest/querying/functions/

[3] https://learn.netdata.cloud/docs/exporting-data/prometheus

[4] https://www.stackhero.io/en/services/Prometheus/documentations/Using-Node-Exporter

[5] Coates, A., and Ng, A.Y. (2012). Learning Feature Representations with K-Means. pp. 561-580.

[6] Neptune Labs. k-means clustering explained. Neptune: a lightweight AI experiment management tool. August 2023. [link]

[7] Islam, M. S., and Miranskyy, A. Anomaly Detection in Cloud Components. 13th IEEE International Conference on Cloud Computing (CLOUD), Beijing, China, 2020, pp. 1-3. [link]

[8] https://github.com/DataCloud-project/ADA-PIPE/tree/main/detect-anomalies

[9] First discrete difference of element. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.diff.html

[10] Provide rolling window calculations. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rolling.html

[11] Conform DataFrame to new index with optional filling logic. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.reindex.html

[12] Pedregosa, F., et al. Scikit-learn: Machine learning in Python. Journal of machine learning research, vol. 12, pp. 2825-2830, 2011.

[13] Google team. Analyzes resource usage and performance characteristics of running containers. https://github.com/google/cadvisor

[14] Prometheus Authors. Monitoring docker container metrics using cadvisor. https://prometheus.io/docs/guides/cadvisor/

[15] Wang, Z., Ma, T., Kong, L., Wen, Z., Li, J., Song, Z., Lu, Y., Chen, G. and Cao, W. (2022). Zero Overhead Monitoring for Cloud-native Infrastructure using {RDMA}. In 2022 USENIX Annual Technical Conference (USENIX ATC 22) (pp. 639-654). [link]

[16] López-Peña, M.A., Díaz, J., Pérez, J.E. and Humanes, H. (2020). DevOps for IoT systems: Fast and continuous monitoring feedback of system availability. IEEE Internet of Things Journal, 7(10), pp.10695-10707. [link]

[17] Luksa, M. (2017). Kubernetes in action. Simon and Schuster.

[18] Rzadca, K., Findeisen, P., Swiderski, J., Zych, P., Broniek, P., Kusmierek, J., Nowak, P., Strack, B., Witusowski, P., Hand., S., & Wilkes, J. (2020, April). Autopilot: workload autoscaling at Google. In Proceedings of the Fifteenth European Conference on Computer Systems (pp. 1-16).

[19] Rossi, F., Cardellini, V., Presti, F. L., & Nardelli, M. (2020). Geo-distributed efficient deployment of containers with Kubernetes. Computer Communications, 159, 161-174.

[20] Toka, L., Dobreff, G., Fodor, B., & Sonkoly, B. (2021). Machine learning-based scaling management for Kubernetes edge clusters. IEEE Transactions on Network and Service Management, 18(1), 958-972.

[21] Ahmad, S., Lavin, A., Purdy, S., & Agha, Z. (2017). Unsupervised real-time anomaly detection for streaming data. Neurocomputing, 262, 134-147.

Co-funded by the Horizon 2020 Framework Programme of the European Union

[22] Lavin, A., and Ahmad, S. Evaluating Real-Time Anomaly Detection Algorithms -- The Numenta Anomaly Benchmark. 14th IEEE International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 2015, pp. 38-44.

[23] Zhang, X., Meng, F., and Xu, J. PerfInsight: A Robust Clustering-Based Abnormal Behavior Detection System for Large-Scale Cloud. 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), San Francisco, CA, USA, 2018, pp. 896-899. [link]

[24] Elarde, J. V., and Brewster, G. B. Performance analysis of application response measurement (ARM) version 2.0 measurement agent software implementations. In Proceedings of the 2000 IEEE International Performance, Computing, and Communications Conference (Cat. No.00CH37086), Phoenix, AZ, USA, 2000, pp. 190-198.

[25] New Relic Python Agent, https://pypi.org/project/newrelic/

[26] Ragunathan, A., Badla, S., & Intuit Inc. Using Cluster Golden Signals to Avoid Alert Fatigue at Scale. Ossna2023. [link]

[27] Agrawal, B., Wiktorski, T. and Rong, C. Adaptive real-time anomaly detection in cloud infrastructures. Concurrency and Computation: Practice and Experience, 29(24), 2017, p.e4193. [link]

[28] Anomaly Advisor - beta launch!!!!, https://community.netdata.cloud/t/anomaly-advisor-beta-launch/2717

Co-funded by the Horizon 2020 Framework Programme of the European Union

APPENDIX A

Anything that is related but not core to the deliverable can go into appendix.

Co-funded by the Horizon 2020
Framework Programme of the European Union