ACCORDION

Adaptive edge/cloud compute and network continuum over a heterogeneous sparse edge infrastructure to support nextgen applications

Deliverable D2.4

Data requirements analysis and collection

ACCORDION receives funding from the European Union's Horizon 2020 2020 research and innovation programme under Grant Agreement No. 871793

DOCUMENT INFORMATION

PROJECT	
PROJECT ACRONYM	ACCORDION
PROJECT FULL NAME	Adaptive edge/cloud compute and network continuum over a heterogeneous sparse edge infrastructure to support nextgen applications
STARTING DATE	01/01/2020 (36 months)
ENDING DATE	31/12/2022
PROJECT WEBSITE	http://www.accordion-project.eu/
TOPIC	ICT-15-2019-2020 Cloud Computing
GRANT AGREEMENT N.	871793
COORDINATOR	CNR
DELIVERABLE INFORMATION	
WORKPACKAGE N. | TITLE	WP 2|Requirements & System Design
WORKPACKAGE LEADER	HUA
DELIVERABLE N. | TITLE	D. 2.4| Data requirements analysis and collection
EDITOR	John Violos (ICCS)
CONTRIBUTOR(S)	Ioannis Korontanis (HUA), Stylianos Tsanakas (ICCS), Hanna Kavalionak (CNR), Vangelis Psomakelis (ICCS), Zinelaabidine Nadir (Aalto), Eduard Marin Fabregas (TID), Saman Zadtootaghaj (TUB), John Violos (ICCS)
REVIEWER	Massimo Coppola (CNR)
CONTRACTUAL DELIVERY DATE	31/12/2020
ACTUAL DELIVERY DATE	31/12/2020
VERSION	1.0
TYPE	Report
DISSEMINATION LEVEL	Public
TOTAL N. PAGES	46
KEYWORDS	Data requirements; data collection

EXECUTIVE SUMMARY

The present document is the result of data requirements analysis and collection. It defines the quality characteristics that good datasets should satisfy and the monitoring sources of data features. This document summarizes all the preparatory work for data collecting that will be used by WP3 Edge infrastructure pool framework, WP4 Edge/Cloud continuum management framework and WP5 Application management framework for experimentation and evaluation.

In section 3, we describe the workflow of dataset construction including data filtering, aggregating and representation. Next, in section 4 we summarize the various data features that have been proposed and used in the literature of Edge and Fog computing in order to put intelligence or enhance the functionality of services like fault tolerance, elasticity, workload balancing, VM migration and task offloading mechanisms. In section 5 we introduce feature selection metrics widely used in the data science field in order evaluate the relation of a candidate data feature with the target value. These feature selection metrics can help us to define the set of features that should be monitored and used by each component.

Section 6 describes the software tools that the ACCORDION components and services require from the infrastructure and application layers in order to collect the data. The main monitoring tool will be Prometheus which uses a pull model to store metrics in a time series database. An additional manual monitoring tool has been developed in python language in order to monitor resource usage in Raspberry pies.

In task 2.4 we had many discussions and intercommunication between partners in order to conclude which tasks of the ACCORDION WP3, WP4 and WP5 have data requirements. Section 7 provides a description of the data features and characteristics for the tasks that have data requirements. In section 8, we conclude the deliverable and give the future direction that the consortium will follow in order to have timely qualitative data for experimentation and evaluation.

DISCLAIMER

REVISION HISTORY LOG

VERSION No.	DATE	AUTHOR(S)	SUMMARY OF CHANGES
0.1	15/03/2020	John Violos (ICCS)	Proposed TOC
0.2	01/10/2020	John Violos (ICCS)	Contributions to sections: 3. Dataset construction process, 4. Data features, 6.2 Manual Monitoring Resources, 6.3 Manual Monitoring Geolocation, 7.5 Resilience Policies & Mechanisms
0.3	01/11/2020	Stylianos Tsanakas (ICCS)	Contributions to sections: 4. Data features, 5. Feature
0.4	23/11/2020	Ioannis Korontanis (HUA) Vangelis Psomakelis (ICCS), Saman Zadtootaghaj (TUB)	selection
0.5	14/12/2020	Hanna Kavalionak (CNR)	Contributions to sections: 6.1 Prometheus Monitoring Tool, 7.2 Data Storage,
0.6	17/12/2020	Zinelaabidine Nadir (AALTO)	7.7 Dynamic QoE
0.7	19/12/2020	John Violos (ICCS)	Assessment
0.8	20/12/2020	John Violos (ICCS)	Contributions to
0.9	22/12/2020	John Violos (ICCS)	section: 7.1 Resource Indexing and Discovery
1.0	31/12/2020	John Violos (ICCS)	Contribution to section: 7.3 Intelligent, Adaptive Resource Orchestration

GLOSSARY

EU	European Union
EC	European Commission
H2020	Horizon 2020 EU Framework Programme for Research and Innovation
QOS	Quality of Service
VM	Virtual Machine
IoT	Internet of Things
QOE	Quality of Experience
DB	Database

TABLE OF CONTENTS

LIST OF TABLES

1 RELEVANCE TO ACCORDION

1.1 PURPOSE OF THIS DOCUMENT

The present document is the result of the collaborative effort of all ACCORDION partners participating in Task 2.4. The objective of this task is to define the data characteristics and the sources of data that require the ACCORDION components and services.

For each component and service that require data we define: the data type for each feature, if it is batch or stream, and the update rates. In addition, we discuss what is an appropriate size of data and last but not least we provide a set of feature selection metrics in order to filter out non-informative or redundant features.

The document offers the methodology in order to collect data from various sources, the available software that monitor the data sources, how to integrate the data, and the preprocessing steps in terms of cleaning and validating.

1.2 RELATION TO OTHER WORKPACKAGES

The task 2.4 Data requirements analysis and collection makes the necessary preparatory work in order to gather the data will be used by the WP3 Edge infrastructure pool framework, the WP4 Edge/Cloud continuum management framework and the WP5 Application management framework.

The WP3, WP4 and WP5 components, tasks, services, models, and any type of analysis that need data in order to provide useful results and decisions should define their data requirements. The data requirements should be derived based on the monitoring applications of ACCORDION project, the edge infrastructure and the deployed applications.

1.3 STRUCTURE OF THE DOCUMENT

In section 2, we introduce the importance of data for decision making in edge fog computing and specifically in the ACCORDION project. In section 3, we describe the workflow of dataset construction from the perspective of data science tailored to the ACCORDION needs. Section 4 provides a long list of data features that have been used in the literature of Edge, Fog and Mist computing. These features are mainly divided into features related to applications and features related to infrastructures as described in the subsections 4.1 and 4.2 respectively. We also mention in subsection 4.3 open data and cost as additional features. In order to evaluate and decide the most important features for each task, we provide in Section 5 feature selection techniques based on the variable types.

Section 6 describes the monitoring tools we can use in order to retrieve data features from infrastructure nodes and applications. Subsection 6.1 gives an overview of Prometheus monitoring tool, subsections 6.2 and 6.3 describe the use of manual monitoring tools. In section 7, we provide for the WP3, WP4 and WP5 tasks a description of their data requirements, specifically Resource indexing and discovery in subsection 71, edge storage in the subsection

7.2, intelligent, adaptive resource orchestration in subsection 7.3, resilience policies and mechanisms in subsection 7.5, privacy preserving mechanisms in subsection 7.6 and dynamic QOE assessment in subsection 7.7. The ACCORDION tasks that do not have data requirements are not included in this list. Section 8 concludes the deliverable.

2 INTRODUCTION

Data are pieces of information that are used in order to work a service and provide useful functionalities. Before the data era, data was just the input and the output of a service while the service could be modeled explicitly by the knowledge experts and the developers. In data driven models, data are also used in order to provide the models of the services. Data analysis and machine learning algorithms derive from models by training datasets. Afterwards, these models are the core of the services.

Edge and cloud computing require data in order to orchestrate and administrate edge nodes, understand the users' behaviors, profile the applications, capture the fluctuation in the workload and estimate the relations between resource usage and deterioration of the Quality of Service (QOS) just to mention a few of the data driven functionalities.

In the ACCORDION project data will not be used only for training models and evaluating their performance. Data can be used for many different uses such as monitoring the connected devices, configuration files, state of Virtual Machines (VM). In this document, we will summarize the data requirements mainly for the WP3, WP4 and WP5. In addition, we will describe all the actions in order to construct quality datasets from raw data and the applications for data gathering.

3 DATASET CONSTRUCTION PROCESS

Dataset construction includes all the necessary actions in order to construct a quality dataset for a component, task, service and analysis. In the rest of the document, we will use the term service to refer to all types of data uses. Different components may share the same dataset or each component may have its own dataset. Generally, speaking, we prefer to build datasets that cover the needs of different components, but most of the time different components have different data requirements and we should avoid putting irrelevant data features together.

The dataset construction process tailored to the needs of Edge computing systems begins with the discussions between the data science expert with the application owners and the infrastructure administrators. These discussions are important to understand what features are need to be monitored for each component. The features can be expressed in categorical (Qualitative) or numeric (Quantitative) types [1]. The categorical data types include nominal which permit permutation of the values and ordinal which preserve an order in the values. The numeric data types include the interval values and the ratio values. In internal values only the differences between values are meaningful, while in Ratio values, both differences and ratios are meaningful.

In the case of training datasets, the amount of data we should gather depends on the complexity of the model and the difficulty of the problem. There is no rule of thumb of the amount of data that is enough but some general directions can be given. We need at least an order of magnitude more examples than trainable parameters in order to avoid the curse of dimensionality. Complex models with high capacity need much more data than simple models. If there are available studies in literature for similar tasks and use cases, we can also target for similar amounts of data.

Data can be gathered from several sources, event monitoring applications such as Prometheus, IoT Sensors open data, historic data files. Data can be in batch or stream format and may have different scales of density among the features or time periods. This heterogeneity of data sources and rates makes data integration a difficult process. The data wrangler should make decisions on how the data features should be aligned. Some general directions are to use timestamps or the IDs of users and devices as key-values to join data records from different sources. The timestamps may not be strict timestamps but a relative time frame in which two or more observations are recorded.

It is a common technique to discard some pieces of information in order to keep a smaller but reliable dataset. When collecting data from several sources, the records can carry mistakes, bad feature values, duplicate or null values. Setting the value ranges or the statistical properties of the values we can detect and filter out erroneous values.

In the case of training predictive models with the dataset, we should pay attention to the skew between training and testing. This means we may have different accuracy at training time than serving time and the reason is not the underfitting or overfitting of the models. The reason may be that the data generators functions in the training process may have different statistical characteristics of the data generator functions in the serving process. This discrepancy in statistical characteristics can be subtle but it may have bad effects on the results. So, we should make sure that the training set is representative of the serving traffic even if it means that we should enrich the training dataset with new observations. A data augmentation process is not always recommended and we should experimentally evaluate if it is applicable.

Data features can be classified into four categories depending on their type and the operations we can run on them. Nominal data refers to attributes whose values can only be distinguished as equal or unequal to each other. They belong to the “Categorical Variables” category and some examples could be “Semantic POI” or “Protocol of Communication”. Another categorical attribute type is ordinal data, whose objects can also be ordered. This includes data such as “Period of day” or “cloud customer satisfaction on a scale from 1-10”. Some operation examples that can be run on nominal and ordinal data respectively are “entropy” and “median”

Aside from Categorical attributes, data can also be numeric. In this category attributes can either be characterized as “Interval” or “Ratio”. In Interval attributes aside from ordering, the difference in values also has a clear meaning, and can be calculated using addition/subtraction. Common examples would be “temperature in Celsius” or “calendar dates”, and operations such as “mean” and “standard deviation” can be used. In Ratio variables, not only the differences but also the ratios have meaning. We can use multiplication/division on the data, with notable examples being “CPU usage” or “RAM usage”. “Geometric mean” and “percent variation” are some of the operations that can be used in this type of data.

4 DATA FEATURES

Different Cloud/Edge stakeholders may have different objectives and need different data features. In this section, we provide an overview of data features used in research publications and technical manuscripts. The purpose of listing the candidate features is to be a reference point to the ACCORDION partners and cloud/edge stakeholders. First, we provide features related to applications and users. Next, we provide features related to infrastructure. In the end, we enlist features that do not belong in the previous two categories. Many of the terms do not have a formal definition and may have different uses in other documents. It is also important to mention that different corporations, institutes and labs may have different definitions for this terminology.

In addition, some features exist in both application and infrastructure lists. This happens because the same term is used in both cases but measures a different quantity. As example the feature of RAM in the context of application means the RAM usage of the application that runs in an edge node. While the feature of RAM for a device means the total RAM usage of the device aggregating the operating system and all the applications that run concurrently. Last but not least, in some cases the characterization of the features as application or infrastructure related is in the eye of the beholder based on the functionalities of the Edge computing components.

4.1 DATA FEATURES RELATED TO APPLICATIONS

Applications may run on devices or offload their tasks to edge nodes and cloud infrastructures following a partial or full offloading policy. In both cases the applications have a set of features that characterize them as a function of the allocated resources and the workload. In the application characterization, we should take into consideration the workload fluctuation, the heterogeneity in edge devices and the dynamic behavior of the edge network. In this section we cover the data features related to applications.

- Response time/latency

  Total time that is taken to respond to a specific application request

- Availability of Application

  It is described as probability i.e., the system is functioning properly after it is requested for use

- Memory usage

  The percentage of RAM memory used by the application compared to the total RAM available on the node

- CPU usage

  Percentage of the time used by the main processor of the node in operations needed by the application. This feature is differentiated based on multiprocessing nodes and multithreading applications

- Number of Disk I/O

  Number of application operations that require reading or writing to disk files on the host device. This feature is differentiated based on the granularity of data blocks in different storage media.

- DB transactions

  Number of application requests made to a database management system. They can be measured either in absolute numbers or as a rate i.e. per second or per minute

- Time duration of Read/Write/Update data transactions

  Percentage of time that the application spends on different types of data transactions

- Transfer time for data transactions

  The total time needed for all the data transactions of the application to be completed

- Size of Transferred File

  Specific file sizes expressed in bytes get transferred by an application through the edge network

- Time-Critical or Mission-Critical Application

  The feature of a software program regarding the requirement to respond within a specific timeframe. If the response is in a batch way it is called time critical application. If the responses are provided continuously in a streaming way it is called mission critical application.

- Location of data (Cloud/Edge/Cloudlet)

  Defines the availability of data shards in different locations of the infrastructure.

- Adaptability

  The ability of all processes to automatically be executed according to different conditions

- Throughput

  The number of tasks whose implementation has been completed successfully per time interval

- Reliability

  The ability for an application to provide accurate or acceptable results in a certain time period

- Usability

  The degree for a user to make use of an application and accomplish his goals with efficiency, effectiveness, and satisfaction

- Other Application KPI

  Other application KPI such as: session length, session interval, number of crashes, number of active users, daily active users, user growth rate, user acquisition, user experience/happiness, average revenue per user

4.2 DATA FEATURES RELATED TO INFRASTRUCTURES

- Availability of Resources

  This refers to the hardware resources (CPU, RAM, Bandwidth, Storage etc) that we have available at any given time

- Type of IoT Devices (i.e., Sensor, Actuator, Computing Units, Gateway)

  An IoT Device is a piece of hardware mostly with a sensor that transmits data from one place to another over the Internet. Types of IoT devices may include wireless sensors, actuators, and processing nodes

- Type of IoT Device Position (Static or Dynamic)

  This refers to whether an IoT Device remains at a fixed geolocation, or it can move (either while staying in the same edge network or not)

- Mobility of devices/users in terms of PoIs or (Lat Lon)

  The geolocation (Latitude/Longitude) or the semantic aspects of a Point of Interests of device

- IoT Device range of connectivity

  The maximum distance coverage that ensures a stable connection between the IoT Device and the edge network

- Energy Consumption

  The amount of energy (in battery percentage or Watt-hours) that the devices use

- The area in which IoT Devices roam or placed

  The bounds and the properties of the area of interest that cover the infrastructure

- Cloud to Edge Data Transfer Vs Edge to Edge Data Transfer, or Edge to Cloudlet Data Transfer

  If we need to prioritize a means of data transfer vs another.

- Computation and Communication Constraints

  All constraints placed by the device's computational capabilities (CPU clock speed, RAM size etc) and the characteristics that define the network and the communication limits

- Protocols That Will Be Used Such as Network Protocols, Wireless Protocols

  A set of standard protocols to be used by the device in correlation with its communication to the edge network

- Incoming Requests

  Any requests to be handled by the device, made by outside sources (edge network)

- Network Data Transfer Speed

  How fast data is transmitted through the edge network

- Transmitting data frequency

  How often do we need to transmit data over the edge network

- Interference Between Applications That Coexist in the Same IoT Device

  Some applications might interfere with each other, e.g. by asking for access to sensors or using up resources at the same time

- Objectives: Performance, Cost: Energy Consumption, Network Delay, Packet Dropouts

  If we have specific objectives to maximize or minimize

- Peculiarities in the Network Structure (Some Nodes Must Be Always Close)

  Specific needs for some use cases regarding network topology

- Constraints in Performance

  Maximum values of device CPU, memory, etc.

- Changes in the Topology of Network Devices

  This refers to changes in the structure of the edge network and the way the devices inside the network are connected to each other

- Time of Deployment (Software or Hardware)

  Time needed to setup the environment (Host, Virtual Machine, Application Container) to be ready for use

- Time of Recoverability

  Time needed for the recovery of the system after a fault occurs

- Any Type of Additional Infrastructure QoS/QoE Metrics

  Ways of measuring Quality of Service or experience (eg time to setup/migrate into a new datacenter)

- Other Infrastructure QoS metrics

  Specific infrastructure metrics that focus on technical network characteristics, such as jittering

- Fault-tolerance Overhead

Determines the total overhead involved while running a fault-tolerance mechanism

4.3 GENERAL DATA FEATURES

- Open data

  Open data is a useful source of information that we can leverage for reasoning and predictive mechanisms. In Edge computing literature, open data such as the weather, temperature and popularity of PoIs [2] have been used for workload modeling.

- Cost

  The description of the system monetarily

5 FEATURE SELECTION

Feature selection is the process of selecting a subset of the available features based on their relevancy to constructing a machine learning or statistical model. The most important reasons for using feature selection techniques include shorter training times, model simplification, and greater generalization. Feature selection techniques are often used in domains where there are many features and comparatively few samples. The following methods help us determine variable correlation in any kind of input and variable type i.e., numerical or categorical.

Pearson's

The Pearson correlation coefficient [3] measures linear correlation between two variables X and Y. The result can indicate positive linear correlation (values close to 1), negative linear correlation (values close to -1 and no linear correlation (values close to 0). The formula used for a population of two variables X and Y is dividing the covariance of X and Y by the standard deviation of these variables multiplied. This method of feature selection applies to problems with numerical inputs and outputs, such as most regression problems.

Spearman's

The Spearman correlation [4] between two variables is equal to the Pearson correlation between the rank values of those two variables. Pearson's correlation only estimates linear relationships, and Spearman's correlation estimates monotonic relationships, regardless of them being linear or non-linear. It is also applicable to numerical input-numerical output problems.

ANOVA

Analysis of variance (ANOVA) [5] is a collection of statistical models and their associated estimation procedures (such as the "variation" among and between groups) used to analyze the differences among group means in a sample. It can determine whether the means of three or more groups are different. ANOVA uses F-tests to statistically test the equality of means. The formula to compute the F-statistic in ANOVA is

F=variation between sample means / variation within the samples

An F-statistic is a ratio of two quantities that are expected to be roughly equal under the null hypothesis, which produces a result of approximately 1. The ANOVA correlation coefficient can be used in Numerical Input/Categorical Output problems, as well as in Categorical Input/Numerical Output problems, but in reverse.

Kendall's

The Kendall rank coefficient [6] is often used to establish whether two variables may be regarded as statistically dependent. It can be used on the same types of problems as the ANOVA correlation coefficient. Kendall rank correlation coefficient computes the difference between the number of concordant and discordant pairs of two variables and divides that by the binomial coefficient.

The result of the Kendall rank coefficient will be in the range of (-1,1), with negative numbers representing a negative dependency, positive numbers representing a dependency, and values that approximate to zero suggesting that the two variables are independent.

Chi-squared

A chi-squared test [7] is a statistical hypothesis test valid to perform when the test statistic is chi-squared distributed under the null hypothesis. Pearson's chi-squared test is used to determine whether there is a statistically significant difference between the expected frequencies and the observed frequencies in one or more categories of a contingency table.

To apply a chi-squared test, we first classify all observations into mutually exclusive classes. Supposing that there are no differences between the classes in the population, the test statistic computed from the observation should follow a “chi-squared” frequency distribution. Then the test will evaluate how likely the observed frequencies would be, if our initial assumption is true.

Mutual Information

The mutual information [8] of two random variables is a measure of mutual dependence between the two variables. It quantifies the amount of information obtained about one random variable through observing the other random variable. Mutual information, as well as Chi-squared test applies to problems with categorical inputs and outputs.

6 MONITORING TOOLS AND SCRIPTS

Edge computing orchestration and decision-making components should have a monitoring system to provide timely and accurate information about the performance of infrastructure, the application behavior and the user behavior. Mostly the monitoring systems include a workload monitoring tool with an application logging mechanism or network sniffers [9] and a resource monitoring tool like Prometheus. In ACCORDION project special emphasis has been given to the resource usage monitoring as a rich source of information that mirrors the status of the whole computing system. Monitoring tools can be available as software tools, ready for use after installation and configuration or manually developed as described in the following subsections.

6.1 PROMETHEUS MONITORING TOOL

In order to gather data, we can use Prometheus a popular open-source monitoring tool which uses a pull model to store metrics in a time series database. The metrics are stored in key value pair format. Prometheus by its own does have a big set of metrics, that's why it uses various exporters to fetch statistics from non-Prometheus systems and convert them into Prometheus metrics. There are various exporters which give different monitoring information, for example:

- node-exporter bare metal metrics for hardware and OS, it needs to be configured in every device that needs to be monitored.
- In case of Kubernetes kube-state-metrics exposes critical metrics about the condition of a Kubernetes cluster, it generates them from the Kubernetes API. It focuses on the health of nodes, pods and deployments. The only requirement for this exporter is to have access to Kubernetes API.
- For Docker container monitoring, cadvisor provides metrics for resource usage and characteristics of the running containers. It needs to be configured in every node that hosts containers.

Prometheus to pull metrics from exporters must configure them as targets with static configuration in case of bare metal or with service discovery in case of Docker Swarm or Kubernetes. In each exporter the metrics can be found in the /metrics endpoint, it is an approach that Prometheus also follows. The metrics that are under /metrics endpoint of Prometheus configured on a K3s (Kubernetes Lightweight) cluster are shown in the below table.To query the time series database Prometheus has its own query language named PromQL, the results can be shown as a graph or as tabular data in Prometheus UI. The data types of PromQL are:

- Instant Vector - a set of time series containing a single sample for each time series, all sharing the same timestamp
- Range Vector - a set of time series containing a range of data points over time for each time series
- Scalar - a simple numeric floating-point value
- String - a simple string value; currently unused

Depending on how the result is shown on as a graph or tabular data some data may not be valid, for example, an expression that returns an instant vector is the only type that can be directly graphed. PromQL also has basic logic, comparing, arithmetic and aggregation operators. For example, to calculate the CPU usage percentage of a by subtracting the idle usage from 100% the query would be:

*100 - (avg by (instance) (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) * 100)*

The irate is a counter to calculate the per-second values, job indicates that node_cpu_seconds_total is a metric from node-exporter, mode returns only the idle cpu metric values and avg aggregates the metric by device with the help of instance and calculates the five minutes average.

Apart from Prometheus UI queries can also performed on the API /api/v1/query on a Prometheus server by adding them as URL parameters. In case of the Prometheus API the response is in JSON format. For successful API requests the returned status code is 2xx, invalid requests return a JSON error object and one of the following HTTP response codes:

- 400 Bad Request when parameters are missing or incorrect.
- 422 Unprocessable Entity when an expression can't be executed (RFC4918).
- 503 Service Unavailable when queries time out or abort.

In the appendix in table 5 are provided all the available data features that can monitor the Prometheus monitoring tool.

6.2 MANUAL MONITORING RESOURCES

Manual monitoring tools can be deployed and run to edge nodes. Specifically, we have developed a python script that combines the psutil [10] and GPUtil [11] libraries and enable us to register metrics about the CPU, RAM, network, HDD and GPU in real time, while the target processes are running. Due to its python nature, it is very lightweight, compatible with edge limitations.

The script allows us to define a set of parameters during each runtime and fine-tuning the monitoring process. These parameters include the snapshot frequency of the metrics watched, the file size limit of the logs in order to more easily process the files, and three lists of excluded devices if we have any, one for each category of devices (HDD, GPU and Network). Table 1 shows an overview of the metrics we are monitoring, we are presenting for each function its name, its return type, its category and a short description:

Function	Type	Category	Description
cpu_{freq}	Float Tuple	CPU	Creates a tuple of the current, minimum and maximum frequency of the CPU.
$cpu_{percent}$	Float Array	CPU	Creates a list of float pointer numbers representing the current system-wide CPU utilization as a percentage for each CPU core at regular intervals.
$disk_{usage}$	Mix Tuple Array	HDD	Creates a tuple including total, used and free space expressed in bytes, plus the percentage usage of each HDD not excluded.
$netI/O_{count}$	JSON Array	Network	Creates a JSON Array containing the following metrics regarding network activity for each not excluded network adapter: • Number of bytes sent • Number of bytes received • Number of packets sent • Number of packets received
$getGPUs$	JSON Array	GPU	Creates a JSON Array containing the following metrics regarding the GPU activity for each not excluded GPU adapter: • Device name • Current load percentage • Current memory utilization • Current memory used in bytes • Current free memory in bytes
$time$	Long Integer	Time	Returns the epoch timestamp for logging.

Table 1 Metrics of Interest of manual monitoring tool

All the data extracted by the monitoring tool are separated by category and converted in JSON format for uniformity and easier usage during the next steps of the modeling process. A JSON schema example is available in table 2.

```
{"timestamp": 1584552696,
"cpu_values": {"cpu_freq": [1400.0, 600.0, 1400.0],
"cpu_percent": [100.0, 0.0, 0.0, 0.0]},
"ram_values": [971055104, 757248000, 22.0, 142651392, 396947456,
308183040, 178515968, 39874560, 391581696, 6840320, 67715072],
"disk_values": [{"device": "/dev/root", "usage": [27666726912,
7450722304, 18787008512, 28.4]},
{"device": "/dev/mmcblk0p6", "usage": [264288256, 54746624,
209541632, 20.7]}],
"network_values": [{"device": "wlan0", "bytes_sent": 0,
"bytes_recv": 0, "packets_sent": 0, "packets_recv": 0},
{"device": "eth0", "bytes_sent": 2821510, "bytes_recv": 31041987,
"packets_sent": 21059, "packets_recv": 32964},
{"device": "lo", "bytes_sent": 0, "bytes_recv": 0,
"packets_sent": 0, "packets_recv": 0}],
"gpu_values": [] }
```

Table 2. JSON of Profiling Features

6.3 MANUAL MONITORING GEOLOCATION

In edge computing, the geolocation of a device in the network is often useful information, especially in cases where the various devices are not in some fixed place, but can move around (such as cellphones, laptops etc). Monitoring the geolocation of such devices may allow us to provide better support against QoS deterioration, or even offer special features, depending on the case.

In that regard, one option is to use IP address location tracking (via some Python packages such as GeoCoder[12] which can easily acquire the approximate geolocation of any device that is connected to the network. The drawback of this option is that the accuracy of the results will not be good enough to determine the exact position, but rather an approximation that could help determine a wider area the device is located in.

Devices that have a GPS sensor installed, can be tracked by using the information the sensor provides, with the use of an API such as Plyer [13], depending on the platform and the device's operating system. Geolocation information acquired via GPS will give much more accurate results and will allow us to monitor the user's movements in greater detail, in order to ensure better Quality of Experience, assisting towards goals such as better connectivity.

7 ASSOCIATION OF COMPONENTS WITH DATA FEATURES AND THEIR CHARACTERISTICS

7.1 RESOURCE INDEXING & DISCOVERY

The aim of “Resource Indexing and Discovery” (RID) component is to keep an up-to-date view on the status of computational resources among the various miniclouds. The data is read from the monitoring functionalities and uses the same data model that is described in the next sections. The data is distributed among the available miniclouds in a way to optimize the efficiency of the retrieval and, at the same time, minimize the disruption if a minicloud becomes unavailable.

In fact, the RID component does not produce new data. Also, the RID service is mostly agnostic about the nature of the data, but its implementation can be configured according to the frequency of its updates. The RID component also provides a service that allows us to extract information about the required resources by running queries on stored data. Such queries can in principle come from any ACCORDION component that wants to find resources with specific computational features or characteristics.

7.2 EDGE STORAGE

The edge storage component is responsible for the management of storage nodes and data storage and transfer on the edge mini-clouds. In order to create optimized policies for these data storage and retrieval tasks the component needs some data related to the resource usage, functionality and infrastructure wellbeing of the storage nodes and the applications that request the data. The component will use these data in order to plan and enact policies concerning the hardware and software architecture, the data storage locations, the data cache and the data access rights.

In order to plan the policies and architectures and create models for real time and predictive corrections and optimizations to these policies we need at least the data described in table 3:

	Type	**Variable**	**Continuous Monitoring**	**Monitoring System**
Node_ID	Nominal	String	✅	Prometheus
Node_I/O	Interval	Float	✅	Prometheus
Node_RAM	Interval	Float	✅	Prometheus
Node_CPU	Interval	Float	✅	Prometheus
Node_Network	Interval	Float	✅	Prometheus

Node_Alive	Interval	Boolean	✅	Prometheus
Node_HDD	Interval	Float	✅	Prometheus

Table 3. Edge Storage Data Requirements

7.3 INTELLIGENT, ADAPTIVE RESOURCE ORCHESTRATION

The Intelligent adaptive resource orchestrator will leverage a resource model and an application model. The resource model will be modeled as a graph and the application model will follow a systematic structure with constraints. In addition, it will take input from data sources of channels and buses that provide information relevant to for static and dynamic features of the two. Until the link of the Intelligent adaptive resource orchestrator with the monitoring section public datasets will be used to estimate the needs of the experimental phase. The data output will be related to orchestrator distributed implementation and they will follow a detailed deployment/reaction plan for each application submitted.

7.4 AI-BASED NETWORK ORCHESTRATION

The aim of this task, task 4.2, is to design and develop a multi-domain AI-based orchestration framework of the network elements by ensuring reliability and latency. This component provides an automated orchestration and intelligent management operations and facilitates the life cycle management of the network slices with the aim of a rapid slice creation and activation, enabling application developers, UC owners (In the case of ACCORDION: ORAMA VR, ORBK and PLEXUS) to define blueprints for their VR/AR ready slices. This component relies on monitoring the compute and network resources at the edge for any potential QoS degradation (e.g., congested links, node capacity excess, etc.) and accordingly predicting the network and computing requirements in real-time that fix these issues and guarantee the application requirements.

This task investigates machine learning techniques and their integration, in order to allow self-configuration and self-optimization capabilities of compute and network resources at the edge. This includes the study of decentralized VR/AR applications, and collect and process various types of data, preparing the ground for intelligent slice deployment. For the training phase, we will rely on our own datasets, collected by deploying either VR/AR applications or use UCs' applications in our testbed. The full spectrum of collected data will be provided in the WP4 and at this stage of the project, we cannot provide these datasets in detail.

7.5 RESILIENCE POLICIES & MECHANISMS

The Resilience policies & mechanisms (RPM) component provides a proactive fault tolerance model using data features related with the resource usage and the mobility. The Mobile aware FT mechanism uses a next position predictor that estimates the geo-location (lat, lon) of a mobile entity in an area of interest in the next time frames. If the mobile entity will be out of the coverage of the Edge infrastructure or there will be a high distribution of mobile entities around a specific Point of interest, then the FT mechanism will predict potential QoS deterioration and trigger proactive measures. The proactive measure will be an intelligent replication (hot/cold migration) that meets the geographical needs of the Edge environment.

The Resource utilization aware FT mechanism will monitor the resources usage that runs on each hybrid edge minicloud in order to reveal at run-time, potential QoS degradation. In case that the deployed resources cannot satisfy the increasing amount of resources usage on the next predicted time steps, then the middleware will trigger mitigation policies such as hot- and cold-migration between neighboring hybrid edge miniclouds and processing edge nodes.

For these two mechanisms there is a need for training and evaluating dataset that can be provided to one or more CSV files (or any equivalent data file). The dataset should contain the features described in the table 4 and the time Interval can be 60 sec.

	Type	**Variable**	**Continuous Monitoring**	**Monitoring System**
Device_ID	Nominal	String	✅	Manual Script
Device_Type	Nominal	String	✅	Manual Script
Device_Latitude	Interval	Float	✅	Manual Script
Device_Longitude	Interval	Float	✅	Manual Script
Timestamp	Ordinal	Float	✅	Manual Script
POI_ID	Nominal	String		File
POI_Semantic Aspects	Composite of Nominals			File
POI_Latitude	Interval	Float		File
POI_Longitude	Interval	Float		File
Edge_ID_CPU_usage	Ratio	Float	✅	Prometheus
Edge_ID_RAM_usage	Ratio	Float	✅	Prometheus

Edge_BW	Ratio	Float	✅	Prometheus
Edge_ID_I/O	Ratio	Float	✅	Prometheus

Table 4. Fault Tolerance Data Requirements

7.6 PRIVACY PRESERVING MECHANISMS

The privacy-preserving component (PPC) is responsible for guaranteeing users' privacy at various levels within the Accordion infrastructure. This component will carry out (at least) three functions. First, PPC will provide the necessary mechanisms to ensure that containers can be correctly executed atop a network infrastructure without the network administrators being able to infer any information about them (e.g., which type of application runs inside them). Second, PPC will enable the generation of machine learning (ML) models that provide high accuracy while at the same time being resistant to attacks that aim to infer private information from the ML model. PPC will also contain privacy-preserving mechanisms suitable for federated learning. Finally, PPC will allow the detection of user data leakage to unauthorised third parties by passively or actively monitoring users on devices and user components (e.g., browsers, containers, etc.).

In order to develop the above-mentioned mechanisms, we require to collect, analyse and process various types of data. In all cases, we will rely on our own datasets. For the privacy-preserving execution of containers, we envision a scenario where containers run in a confidential and isolated manner protected by a trusted execution environment (TEE). However, this approach does not protect the interactions between the container and the host kernel, which, based on our hypothesis, could become a unique fingerprint for containers. To test our hypothesis and eliminate the fingerprint (if it exists), we will crawl the Docker hub website in order to obtain the names of many popular container images, then run each of the container images separately in order to collect and analyse their syscalls patterns. Similarly, to detect user data leakage, we will collect the data exchanged between different entities and analyse the obtained information at distinct levels (e.g., in the application or network layer). This includes plaintext and encrypted data collected while users browse the Web or when containers communicate with each other (among others). Finally, we envisage the design and implementation of generic techniques -- that can be combined with other technologies such as TEEs -- to produce privacy-preserving ML models that provide high accuracy without incurring a high overhead. The proposed techniques can then be adjusted to fit the needs of a specific ML task within the Accordion infrastructure.

7.7 DYNAMIC QOE ASSESSMENT

For quality assessment task, the data will be collected once during the development of model through subjective experiment, and once the model is developed and employed in the accordion framework. For the training phase, subjective data will be collected according to the ethic policy at the host institute. The data that is collected is subjective ratings in scale of 1 to 7 which is the judgement of users about the quality features of service or application that the ser experienced. In addition, some demographic information will be collected prior to the experiment such as age, gender, and level of experience to the test service or application. It has to be noted that the consent form to collect the data for research purposes will be given to the participant before the start of the test. It has to be noted that no personal information that can link the participants to their collected data will be gathered, e.g., name or phone number.

For testing the model in the accordion framework, depends on the application, network parameters, encoding parameters, and client information might be collected. The full spectrum of collected data will be provided in the WP6 report and at this stage of the project cannot be determined in detail.

8 CONCLUSIONS

In the report of the data requirements analysis and collection, we discussed the importance of quality data for timely and effective decision making and orchestration in Edge computing infrastructures. Deep learning and Artificial Intelligence mechanisms are data driven models that provide accurate results based on the quality of their training data. So, we described the steps that we should follow in order to construct the appropriate datasets for each component, how to filter the important data features and discard the features that are not related to the target of the models.

In the context of Edge computing systems, we presented the monitoring tools which we can use in the ACCORDION project and specifically the Prometheus monitoring tool. The monitoring tools are the valuable sources that we can use to record the features of interest. In the last section we presented the ACCORDION components that have data requirements and the description of the data they need.

REFERENCES

[1] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Pearson, 2006.

[2] J. Violos, S. Pelekis, A. Berdelis, S. Tsanakas, K. Tserpes, and T. Varvarigou, “Predicting Visitor Distribution for Large Events in Smart Cities,” in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Feb. 2019, pp. 1–8, doi: 10.1109/BIGCOMP.2019.8679181.

[3] J. Benesty, J. Chen, Y. Huang, and I. Cohen, “Pearson Correlation Coefficient,” in *Noise Reduction in Speech Processing*, I. Cohen, Y. Huang, J. Chen, and J. Benesty, Eds. Berlin, Heidelberg: Springer, 2009, pp. 1–4.

[4] C. Wissler, “The Spearman Correlation Formula,” *Science*, vol. 22, no. 558, pp. 309–311, 1905.

[5] L. Stehle and S. Wold, “Analysis of variance (ANOVA),” *Chemometrics and Intelligent Laboratory Systems*, vol. 6, no. 4, pp. 259–272, Nov. 1989, doi: 10.1016/0169-7439(89)80095-4.

[6] M. G. Kendall, “Partial Rank Correlation,” *Biometrika*, vol. 32, no. 3/4, pp. 277–283, 1942, doi: 10.2307/2332130.

[7] P. E. Greenwood and M. S. Nikulin, *A Guide to Chi-Squared Testing*. John Wiley & Sons, 1996.

[8] B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, “Mutual Information Analysis,” in *Cryptographic Hardware and Embedded Systems – CHES 2008*, Berlin, Heidelberg, 2008, pp. 426–442, doi: 10.1007/978-3-540-85053-3_27.

[9] M. C. Calzarossa, L. Massari, and D. Tessera, “Workload Characterization: A Survey Revisited,” *ACM Comput. Surv.*, vol. 48, no. 3, p. 48:1–48:43, Feb. 2016, doi: 10.1145/2856127.

[10] GitHub, G. Rodola’, giampaolo/psutil, https://github.com/giampaolo/psutil. Last accessed 5 Jun 2020

[11] GitHub, A. K. Mortensen, anderskm/gputil https://github.com/anderskm/gputil. Last accessed 5 Jun 2020

[12] “Geocoder: Simple, Consistent — geocoder 1.38.1 documentation.” https://geocoder.readthedocs.io/ (accessed Dec. 22, 2020).

[13] K. team, *plyer: Platform-independent wrapper for platform-dependent APIs*. https://pypi.org/project/plyer/ (accessed Dec. 22, 2020).

APPENDIX A. PROMETHEUS METRICS

METRICS	SUMMARY
go_gc_duration_seconds	A summary of the pause duration of garbage collection cycles.
go_goroutines	Number of goroutines that currently exist.
go_info	Information about the Go environment.
go_memstats_alloc_bytes	Number of bytes allocated and still in use.
go_memstats_alloc_bytes_total	Total number of bytes allocated, even if freed.
go_memstats_buck_hash_sys_bytes	Number of bytes used by the profiling bucket hash table.
go_memstats_frees_total	Total number of frees.
go_memstats_gc_cpu_fraction	The fraction of this program's available CPU time used by the GC since the program started.
go_memstats_gc_sys_bytes	Number of bytes used for garbage collection system metadata.
go_memstats_heap_alloc_bytes	Number of heap bytes allocated and still in use.
go_memstats_heap_idle_bytes	Number of heap bytes waiting to be used.
go_memstats_heap_inuse_bytes	Number of heap bytes that are in use.
go_memstats_heap_objects	Number of allocated objects.
go_memstats_heap_released_bytes	Number of heap bytes released to OS.
go_memstats_heap_sys_bytes	Number of heap bytes obtained from system.
go_memstats_last_gc_time_seconds	Number of seconds since 1970 of last garbage collection.
go_memstats_lookups_total	Total number of pointer lookups.
go_memstats_mallocs_total	Total number of mallocs.

go_memstats_mcache_inuse_bytes	Number of bytes in use by mcache structures.
go_memstats_mcache_sys_bytes	Number of bytes used for mcache structures obtained from system.
go_memstats_mspan_inuse_bytes	Number of bytes in use by mspan structures.
go_memstats_mspan_sys_bytes	Number of bytes used for mspan structures obtained from system.
go_memstats_next_gc_bytes	Number of heap bytes when next garbage collection will take place.
go_memstats_other_sys_bytes	Number of bytes used for other system allocations.
go_memstats_stack_inuse_bytes	Number of bytes in use by the stack allocator.
go_memstats_stack_sys_bytes	Number of bytes obtained from system for stack allocator.
go_memstats_sys_bytes	Number of bytes obtained from system.
go_threads	Number of OS threads created.
net_conntrack_dialer_conn_attempted_total	Total number of connections attempted by the given dialer a given name.
net_conntrack_dialer_conn_closed_total	Total number of connections closed which originated from the dialer of a given name.
net_conntrack_dialer_conn_established_total	Total number of connections successfully established by the given dialer a given name.
net_conntrack_dialer_conn_failed_total	Total number of connections failed to dial by the dialer a given name.
net_conntrack_listener_conn_accepted_total	Total number of connections opened to the listener of a given name.
net_conntrack_listener_conn_closed_total	Total number of connections closed that were made to the listener of a given name.
process_cpu_seconds_total	Total user and system CPU time spent in seconds.

process_max_fds	Maximum number of open file descriptors.
process_open_fds	Number of open file descriptors.
process_resident_memory_bytes	Resident memory size in bytes.
process_start_time_seconds	Start time of the process since unix epoch in seconds.
process_virtual_memory_bytes	Virtual memory size in bytes.
process_virtual_memory_max_bytes	Maximum amount of virtual memory available in bytes.
prometheus_api_remote_read_queries	The current number of remote read queries being executed or waiting.
prometheus_build_info	A metric with a constant '1' value labeled by version, revision, branch, and goversion from which prometheus was built.
prometheus_config_last_reload_success_timestamp_seconds	Timestamp of the last successful configuration reload.
prometheus_config_last_reload_successful	Whether the last configuration reload attempt was successful.
prometheus_engine_queries	The current number of queries being executed or waiting.
prometheus_engine_queries_concurrent_max	The max number of concurrent queries.
prometheus_engine_query_duration_seconds	Query timings
prometheus_engine_query_log_enabled	State of the query log.
prometheus_engine_query_log_failures_total	The number of query log failures.
prometheus_http_request_duration_seconds	Histogram of latencies for HTTP requests.
prometheus_http_requests_total	Counter of HTTP requests.
prometheus_http_response_size_bytes	Histogram of response size for HTTP requests.
prometheus_notifications_alertmanagers_discovered	The number of alertmanagers discovered and active.
prometheus_notifications_dropped_total	Total number of alerts dropped due to errors when sending to Alertmanager.

prometheus_notifications_errors_total	Total number of errors sending alert notifications.
prometheus_notifications_latency_seconds	Latency quantiles for sending alert notifications.
prometheus_notifications_queue_capacity	The capacity of the alert notifications queue.
prometheus_notifications_queue_length	The number of alert notifications in the queue.
prometheus_notifications_sent_total	Total number of alerts sent.
prometheus_remote_storage_highest_timestamp_in_seconds	Highest timestamp that has come into the remote storage via the Appender interface, in seconds since epoch.
prometheus_remote_storage_samples_in_total	Samples in to remote storage, compare to samples out for queue managers.
prometheus_remote_storage_string_interner_zero_reference_releases_total	The number of times release has been called for strings that are not interned.
prometheus_rule_evaluation_duration_seconds	The duration for a rule to execute.
prometheus_rule_evaluation_failures_total	The total number of rule evaluation failures.
prometheus_rule_evaluations_total	The total number of rule evaluations.
prometheus_rule_group_duration_seconds	The duration of rule group evaluations.
prometheus_rule_group_interval_seconds	The interval of a rule group.
prometheus_rule_group_iterations_missed_total	The total number of rule group evaluations missed due to slow rule group evaluation.
prometheus_rule_group_iterations_total	The total number of scheduled rule group evaluations, whether executed or missed.
prometheus_rule_group_last_duration_seconds	The duration of the last rule group evaluation.
prometheus_rule_group_last_evaluation_timestamp_seconds	The timestamp of the last rule group evaluation in seconds.
prometheus_rule_group_rules	The number of rules.
prometheus_sd_consul_rpc_duration_seconds	The duration of a Consul RPC call in seconds.

prometheus_sd_consul_rpc_failures_total	The number of Consul RPC call failures.
prometheus_sd_discovered_targets	Current number of discovered targets.
prometheus_sd_dns_lookup_failures_total	The number of DNS-SD lookup failures.
prometheus_sd_dns_lookups_total	The number of DNS-SD lookups.
prometheus_sd_failed_configs	Current number of service discovery configurations that failed to load.
prometheus_sd_file_read_errors_total	The number of File-SD read errors.
prometheus_sd_file_scan_duration_seconds	The duration of the File-SD scan in seconds.
prometheus_sd_kubernetes_events_total	The number of Kubernetes events handled.
prometheus_sd_kubernetes_http_request_duration_seconds	Summary of latencies for HTTP requests to the Kubernetes API by endpoint.
prometheus_sd_kubernetes_http_request_total	Total number of HTTP requests to the Kubernetes API by status code.
prometheus_sd_kubernetes_workqueue_depth	Current depth of the work queue.
prometheus_sd_kubernetes_workqueue_items_total	Total number of items added to the work queue.
prometheus_sd_kubernetes_workqueue_latency_seconds	How long an item stays in the work queue.
prometheus_sd_kubernetes_workqueue_longest_running_processor_seconds	Duration of the longest running processor in the work queue.
prometheus_sd_kubernetes_workqueue_unfinished_work_seconds	How long an item has remained unfinished in the work queue.
prometheus_sd_kubernetes_workqueue_work_duration_seconds	How long processing an item from the work queue takes.
prometheus_sd_received_updates_total	Total number of update events received from the SD providers.
prometheus_sd_updates_delayed_total	Total number of update events that couldn't be sent immediately.
prometheus_sd_updates_total	Total number of update events sent to the SD consumers.

prometheus_target_interval_length_seconds	Actual intervals between scrapes.
prometheus_target_metadata_cache_bytes	The number of bytes that are currently used for storing metric metadata in the cache
prometheus_target_metadata_cache_entries	Total number of metric metadata entries in the cache
prometheus_target_scrape_pool_reloads_failed_total	Total number of failed scrape loop reloads.
prometheus_target_scrape_pool_reloads_total	Total number of scrape loop reloads.
prometheus_target_scrape_pool_sync_total	Total number of syncs that were executed on a scrape pool.
prometheus_target_scrape_pools_failed_total	Total number of scrape pool creations that failed.
prometheus_target_scrape_pools_total	Total number of scrape pool creation attempts.
prometheus_target_scrapes_cache_flush_forced_total	How many times a scrape cache was flushed due to getting big while scrapes are failing.
prometheus_target_scrapes_exceeded_sample_limit_total	Total number of scrapes that hit the sample limit and were rejected.
prometheus_target_scrapes_sample_duplicate_timestamp_total	Total number of samples rejected due to duplicate timestamps but different values
prometheus_target_scrapes_sample_out_of_bounds_total	Total number of samples rejected due to timestamp falling outside of the time bounds
prometheus_target_scrapes_sample_out_of_order_total	Total number of samples rejected due to not being out of the expected order
prometheus_target_sync_length_seconds	Actual interval to sync the scrape pool.
prometheus_template_text_expansion_failures_total	The total number of template text expansion failures.
prometheus_template_text_expansions_total	The total number of template text expansions.
prometheus_treecache_watcher_goroutines	The current number of watcher goroutines.

prometheus_treecache_zookeeper_failures_total	The total number of ZooKeeper failures.
prometheus_tsdb_blocks_loaded	Number of currently loaded data blocks
prometheus_tsdb_checkpoint_creations_failed_total	Total number of checkpoint creations that failed.
prometheus_tsdb_checkpoint_creations_total	Total number of checkpoint creations attempted.
prometheus_tsdb_checkpoint_deletions_failed_total	Total number of checkpoint deletions that failed.
prometheus_tsdb_checkpoint_deletions_total	Total number of checkpoint deletions attempted.
prometheus_tsdb_compaction_chunk_range_seconds	Final time range of chunks on their first compaction
prometheus_tsdb_compaction_chunk_samples	Final number of samples on their first compaction
prometheus_tsdb_compaction_chunk_size_bytes	Final size of chunks on their first compaction
prometheus_tsdb_compaction_duration_seconds	Duration of compaction runs
prometheus_tsdb_compaction_populating_block	Set to 1 when a block is currently being written to the disk.
prometheus_tsdb_compactions_failed_total	Total number of compactions that failed for the partition.
prometheus_tsdb_compactions_skipped_total	Total number of skipped compactions due to disabled auto compaction.
prometheus_tsdb_compactions_total	Total number of compactions that were executed for the partition.
prometheus_tsdb_compactions_triggered_total	Total number of triggered compactions for the partition.
prometheus_tsdb_head_active_appenders	Number of currently active appender transactions
prometheus_tsdb_head_chunks	Total number of chunks in the head block.
prometheus_tsdb_head_chunks_created_total	Total number of chunks created in the head
prometheus_tsdb_head_chunks_removed_total	Total number of chunks removed in the head

prometheus_tsdb_head_gc_duration_seconds	Runtime of garbage collection in the head block.
prometheus_tsdb_head_max_time	Maximum timestamp of the head block. The unit is decided by the library consumer.
prometheus_tsdb_head_max_time_seconds	Maximum timestamp of the head block.
prometheus_tsdb_head_min_time	Minimum time bound of the head block. The unit is decided by the library consumer.
prometheus_tsdb_head_min_time_seconds	Minimum time bound of the head block.
prometheus_tsdb_head_samples_appended_total	Total number of appended samples.
prometheus_tsdb_head_series	Total number of series in the head block.
prometheus_tsdb_head_series_created_total	Total number of series created in the head
prometheus_tsdb_head_series_not_found_total	Total number of requests for series that were not found.
prometheus_tsdb_head_series_removed_total	Total number of series removed in the head
prometheus_tsdb_head_truncations_failed_total	Total number of head truncations that failed.
prometheus_tsdb_head_truncations_total	Total number of head truncations attempted.
prometheus_tsdb_isolation_high_watermark	The highest TSDB append ID that has been given out.
prometheus_tsdb_isolation_low_watermark	The lowest TSDB append ID that is still referenced.
prometheus_tsdb_lowest_timestamp	Lowest timestamp value stored in the database. The unit is decided by the library consumer.
prometheus_tsdb_lowest_timestamp_seconds	Lowest timestamp value stored in the database.
prometheus_tsdb_mmap_chunk_corruptions_total	Total number of memory-mapped chunk corruptions.
prometheus_tsdb_out_of_bound_samples_total	Total number of out of bound samples ingestion failed attempts.

prometheus_tsdb_out_of_order_samples_total	Total number of out of order samples ingestion failed attempts.
prometheus_tsdb_reloads_failures_total	Number of times the database failed to reload block data from disk.
prometheus_tsdb_reloads_total	Number of times the database reloaded block data from disk.
prometheus_tsdb_retention_limit_bytes	Max number of bytes to be retained in the tsdb blocks, configured 0 means disabled
prometheus_tsdb_size_retentions_total	The number of times that blocks were deleted because the maximum number of bytes was exceeded.
prometheus_tsdb_storage_blocks_bytes	The number of bytes that are currently used for local storage by all blocks.
prometheus_tsdb_symbol_table_size_bytes	Size of symbol table on disk (in bytes)
prometheus_tsdb_time_retentions_total	The number of times that blocks were deleted because the maximum time limit was exceeded.
prometheus_tsdb_tombstone_cleanup_seconds	The time taken to recompact blocks to remove tombstones.
prometheus_tsdb_vertical_compactions_total	Total number of compactions done on overlapping blocks.
prometheus_tsdb_wal_completed_pages_total	Total number of completed pages.
prometheus_tsdb_wal_corruptions_total	Total number of WAL corruptions.
prometheus_tsdb_wal_fsync_duration_seconds	Duration of WAL fsync.
prometheus_tsdb_wal_page_flushes_total	Total number of page flushes.
prometheus_tsdb_wal_segment_current	WAL segment index that TSDB is currently writing to.
prometheus_tsdb_wal_truncate_duration_seconds	Duration of WAL truncation.
prometheus_tsdb_wal_truncations_failed_total	Total number of WAL truncations that failed.
prometheus_tsdb_wal_truncations_total	Total number of WAL truncations attempted.

prometheus_tsdb_wal_writes_failed_total	Total number of WAL writes that failed.
prometheus_web_federation_errors_total	Total number of errors that occurred while sending federation responses.
prometheus_web_federation_warnings_total	Total number of warnings that occurred while sending federation responses.
promhttp_metric_handler_requests_in_flight	Current number of scrapes being served.
promhttp_metric_handler_requests_total	Total number of scrapes by HTTP status code.

Table 5. Prometheus Metrics