CEN

WORKSHOP

AGREEMENT

CWA 17494

January 2020

ICS 35.080

English version

Analytics Insights and Scaling Policies for Microservices

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

Ref. No.:CWA 17494:2020 E

Contents Page

Foreword

This CEN Workshop Agreement has been developed in accordance with the CEN-CENELEC Guide 29 "CEN/CENELEC Workshop Agreements – The way to rapid consensus" and with the relevant provisions of CEN/CENELEC Internal Regulations - Part 2. It was approved by a Workshop of representatives of interested parties on 2019-10-25, the constitution of which was supported by CEN following the public call for participation made on 2018-09-10. However, this CEN Workshop Agreement does not necessarily include all relevant stakeholders.

The final text of this CEN Workshop Agreement was provided to CEN for publication on 2019-12-16.

The following organizations and individuals developed and approved this CEN Workshop Agreement:

- CAS Software AG/Spiros Alexakis
- Julia Vuong/CAS Software AG
- Manos Papoutsakis/FORTH
- Marios Phinikettos/Suite 5
- Sotirios Koussouris/Suite 5
- Pedro Sanchez/Technical University of Cartagena
- Diego Alonso/Technical University of Cartagena
- Ioannis Ledakis/UBITECH
- Panos Parthenis/UBITECH
- Panagiotis Gouvas/UBITECH
- Demetris Trihinas/University of Cyprus
- George Pallis/University of Cyprus
- Marios Dikaiakos/University of Cyprus
- Thanasis Tryfonos/University of Cyprus

Attention is drawn to the possibility that some elements of this document may be subject to patent rights. CEN-CENELEC policy on patent rights is described in CEN-CENELEC Guide 8 "Guidelines for Implementation of the Common IPR Policy on Patent". CEN shall not be held responsible for identifying any or all such patent rights.

Although the Workshop parties have made every effort to ensure the reliability and accuracy of technical and non-technical descriptions, the Workshop is not able to guarantee, explicitly or implicitly, the correctness of this document. Anyone who applies this CEN Workshop Agreement shall be aware that neither the Workshop, nor CEN, can be held liable for damages or losses of any kind whatsoever. The use of this CEN Workshop Agreement does not relieve users of their responsibility for their own actions, and they apply this document at their own risk.

Introduction

The emergence of cloud computing altered radically the way modern applications are managed. Virtualization offers many technical and financial advantages since it contributes to rapid provisioning and to decrease of operational expenses. One of the most significant implications of the cloud computing dominance is the emergence of microservices as the de-facto application development paradigm. Thus, modern applications are not architected in a monolithic way. Instead, applications are decomposed in several microservices that can be managed independently. Management refers to all states of a microservice e.g. start, stop, scale in, scale out etc. One of the crucial aspects of microservice management is elasticity. Elasticity refers to the way a microservice is reacting to the increase or decrease of its load. Microservices that have the ability to scale in/out are considered elastic-by-design. The scope of this document is to set the guidelines for platform agnostic elasticity management.

1 Scope

This CEN Workshop Agreement gives guidelines for platform-agnostic elasticity management of elastic-by-design microservices. Platform-agnostic implies that the mechanism/orchestration entity which will perform the actual scale-in/out process is outside of the scope of this document. Instead, the definition of the actual elasticity events and the relationship of this definition with the underlying monitoring mechanisms will be formally described. The specification is using the Backus Naur form [1].

This document is applicable to independent software vendors (also known as ISVs) or developers of microservice orchestration platforms.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at http://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
Extended Backus–Naur form
EBNF
formal notation which can be used to express a context-free grammar, consisting of terminal symbols and non-terminal production rules which are the restrictions governing how terminal symbols can be combined into a legal sequence

3.2
Infrastructure as a Service
IaaS
form of cloud computing that provides virtualized computing resources over the internet to provision processing, storage, networks, and other fundamental computing resources

3.3
Independent Software Vendor
ISV
software producer that is not owned or controlled by a hardware manufacturer or a company whose primary function is to distribute software

4 Metric Definition

A Metric is an essential part of an autonomous system that aims to capture and quantify the behaviour of a system's element (e.g. how many requests are handled by a service). In Table 1, the definition of a *Metric* is described. The unique identifier of a metric is composed by the name of the metric and the identifier of the agent that belongs to; each agent is responsible for collecting one or more metrics through its probes. The *MetricDefinition* production rule consists of useful information about the metric, such as, description, type, unit, group and an initial value. The attribute *HigherIsBetter* is useful for visualization and analysis purposes, as it specifies whether a higher value is better or not in terms of

quality. The *MetricConfiguration* production rule, allows the use of configurations, such as controlling the intrusiveness and accuracy of various adaptive monitoring techniques.

Table 1 — Metric EBNF Definition

Metric	::=	MetricInstanceID ":" MetricDefinition "," MetricConfiguration
MetricInstanceID	::=	"(" MetricName "," AgentID ")"
MetricName	::=	String
AgentID	::=	String
MetricDefinition	::=	Description Type Unit InitValue Min Max Group HigherIsBetter
Description	::=	String
Type	::=	"Float" | "Double" | "Integer" | "Boolean" | "String"
Unit	::=	String
InitValue	::=	Number | String
Number	::=	[+-]? ([0-9]*[.])? [0-9]+
Min	::=	Number | String
Max	::=	Number | String
Group	::=	String
HigherIsBetter	::=	"TRUE" | "FALSE"
MetricConfiguration	::=	Param ("," Param)*
Param	::=	Key Value
Key	::=	String
Value	::=	"Object"

5 Measurement Extraction

A Monitoring Agent is composed from a set of probes and tags. Table 2, describes the definition of the Monitoring Agent. The tags are useful for grouping agents that belong to the same tier (e.g. business-tier or presentation-tier) or services that belong to a specific zone (e.g. eu_west, eu_central). Each probe contains a set of metrics that are collected based on a periodical cycle (e.g. every 5 s) or when certain conditions are satisfied (event based). With the use of conditions, for example, an agent collects measurements only when the value is above or below a certain threshold (e.g. memory_usage < 5), or there is a change on its statistical properties (e.g. DIFF (cpu_usage) > = 2). The DIFF function denotes the difference between the value and the arithmetic mean of the timeseries stream in standard deviation units. The APTITUDE denotes the magnitude of change of the value since the last measurement.

Table 2 — Monitoring Agent EBNF Definition

Agent	::=	AgentID ":" Probes "," Tags
AgentID	::=	UUID
UUID	::=	String
Probes	::=	Probe ("," Probe)*

Probe	::=	Name "," Metrics "," CollectionEvent
Metrics	::=	Metric ("," Metric)*
CollectionEvent	::=	Periodical | EventBased
Periodical	::=	"COLLECT EVERY" PositiveInteger TimeUnit
PositiveInteger	::=	[1-9]([0-9])*
TimeUnit	::=	"Milliseconds" | "Seconds"| "Minutes"| "Hours"
EventBased	::=	"WHEN" MetricConditions
MetricConditions	::=	MetricCondition ("AND" MetricCondition)*
MetricCondition	::=	MetricFunction "(" MetricName ")" RelOp Number
MetricFunction	::=	"LAST_VALUE" | "DIFF" | "APTITUDE"
RelOp	::=	"LT" | "GT" | "EQ" | "NEQ" | "GTE" | "LTE"
Number	::=	[+-]? ([0-9]*[.])? [0-9]+

6 Analytics Insight Definition Language

An Analytics Insight is basically a composite metric, namely one that is composed from multiple metrics, via composition and/or aggregation. A user can define an insight by firstly specifying the observation period of her interest and then compose with basic arithmetic operations the aggregated metric values exposed by a set of members. The Membership production rule allows the user to specify from which agents the metrics are collected.

A CompositeExpr is either a simple expression, denoted as Expr, or is recursievely constructed via left and right-hand composite expressions operated by a binary operation (e.g. ADD, DIV). An Expr can be an Aggregate function (e.g. MEDIAN (cpu_usage)), a MetricStream or a Number. Optionally, a Filter can be attached to an Expr so that left-hand operations are only processed if the filter predicate evaluates to true. As such, a Filter is composed from applying on a metric a relational operation and a CompositeExpr, with users also able to concatenate multiple filters with the AND logical operator. In turn, an Aggregate is either window-based or accumulated function. The difference is in the application of the aggregate on the metric stream. For window-based aggregates (WindowedFunc), a window is needed to denote the time interval of interest for aggregating values, while for accumulated aggregates (AccumFunc) the result is computed solely based on previous values. A WindowedFunc can be either a UserFunc or a PrimitiveWindowedFunc. A UserFunc is a user-defined function that takes: i) a list of key-value pairs as parameters; ii) a MetricStream as input; and produce another MetricStream. Respectively, an AccumFunc can be either a UserFunc or a PrimitiveAccumFunc. A PrimitiveWindowedFunc is a primitive windowed function like AVERAGE, SUM, and COUNT, while a PrimitiveAccumFunc is a primitive accumulated function like RUNNING_SUM, RUNNING_MEAN, etc.

Both windowed and accumulated functions require as input a MetricStream, which is composed from a list of Metrics and optionally a Membership description. In turn, the membership description is used when metrics are compiled as aggregates from multiple monitoring sources. When the membership is omitted, measurements satisfying the metric description from all monitoring sources are considered in the insight computation. What is more, Filters can be attached and applied directly to the metric stream. Furthermore, in the composition definition, an optional BY statement is available which permits grouping the expression evaluation on a given Metric key.

Finally, the optional WITH statement allows the user to define certain optimization strategies and constraints to improve runtime performance. Specifically, SALIENCE denotes the importance of an insight, allowing the user to prioritize query execution over other queries. In turn, SAMPLE allows users

to specify that query execution can be applied on a percentage of the available measurements so that an approximate answer is given in a fraction of the time required to execute the query on the entire data. The language also supports users to set the MAX ERROR and CONFIDENCE which are optimization constraints allowing for the query to be executed on a sample of the data, where the constructed sample must satisfy the aforementioned constraints.

Table 3 — Insight EBNF Definition

Insight	::=	InsightID ":" Composition "EVERY" Window ["WITH" Optimizations]
InsightID	::=	String
Composition	::=	CompositeExpr ["BY" MetricName]
CompositeExpr	::=	CompositeExpr BinOp CompositeExpr | MapOp "(" Expr ")" | Expr
BinOp	::=	"+" | "-" | "*" | "/" | "DIV" | "MOD"
Expr	::=	Aggregate ["WHEN" Filters] | MetricStream | Number
Aggregate	::=	WindowedFunc | AccumFunc
WindowedFunc	::=	(UserFunc | PrimitiveWindowedFunc) "(" MetricStream "," Window ")"
UserWindowedFunc	::=	FuncName [Params]
FuncName	::=	String
Params	::=	"[" Key "=" Value ("," Key "=" Value)* "]"
Key	::=	String
Value	::=	String | Number
PrimitiveWindowedFunc	::=	"SUM" | "COUNT" | "PRODUCT" | "ARITHMETIC_MEAN" | "GEOMETRIC_MEAN" | "HARMONIC_MEAN" | "MIN" | "MAX" | "MEDIAN" | "PERENTILE" "[" Percent "]" | "TOP_K" "[" PositiveInt "]"
UserWindowedFunc	::=	FuncName [Params]
FuncName	::=	String
Params	::=	Key "=" Value ("," Key "=" Value)*
AccumFunc		(UserFunc | PrimitiveAccumFunc) "(" MetricStream ")"
PrimitiveAccumFunc	::=	"RUNNING_SDEV" | "RUNNING_MEAN" | "RUNNING_MAX" | "RUNNING_MIN" | EWMA "[" Percent "]" | PEWMA "[" Percent "]"
MetricStream	::=	Metrics ["FROM" Membership] ["WHEN" Filters]
Metrics	::=	MetricName ("," MetricName)*
Membership	::=	AgentID (","AgentID)* | "TAG" "=" Tag

Filters	::=	Filters (“AND” Filter)*
Filter	::=	MetricName RelOp CompositeExpr
RelOp	::=	“LT” | “GT” | “EQ” | “NEQ” | “GTE” | “LTE”
Window	::=	TimePeriod
TimePeriod	::=	PositiveInt TimeUnit
TimeUnit	::=	“MILLIS” | “SECONDS” | “MINUTES” | “HOURS”
Tag	::=	String
Optimizations	::=	[“SALIENCE” PositiveInt] | [(“MAX_ERROR” Percent “AND” “CONFIDENCE” Percent | “SAMPLE” Percent)]
MapOp	::=	PrimitiveMapOp | UserFunc
PrimitiveMapOp	::=	“ABS” | “SQR” | “SQRT”

7 Elasticity Rule Definition and Evaluation

An Elasticity Policy, depicted in Table 4, is composed by an elasticity trigger and an elasticity action. The policy specifies the action to be enforced when the policy is triggered. Note that, in case of conflicting policies, the priority value is used to resolve the conflict. An example of such policy is shown in Figure 1. In this example, the policy specifies that one more service of type svc-s (streaming service) should be added when there is i) a sufficient increase (10 requests/sec in that last 5 minutes) on the number of requests in the European region; ii) the aggregated cost of these services do not exceed the budget constraint (1,5 credits/hour); iii) and also there are less than 10 services running.

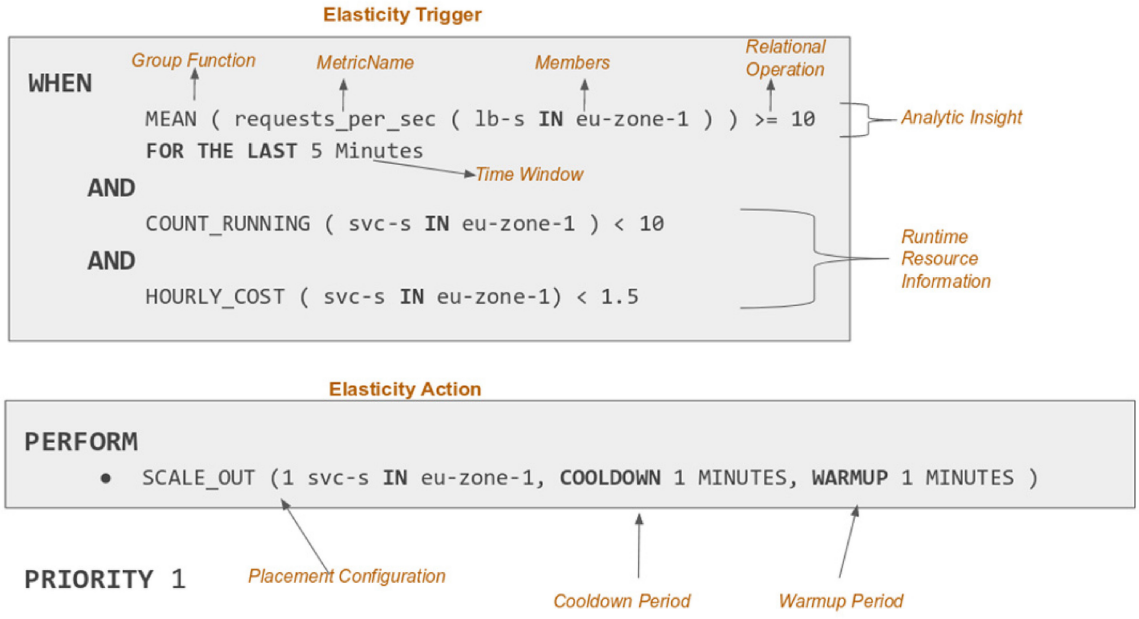

Figure 1 — Elasticity Policy Example

A trigger, as described in Table 5, may contain multiple elasticity conditions that should all be satisfied so that the policy is triggered. Each condition can be either a runtime information expression or a scheduled event. The Runtime Information expression can be composed from an existing insight metric or from an aggregated runtime information from resources, such as running instances, or hourly cost.

The value of the runtime information is then compared with a user-defined value, in order to specify when the condition should evaluate to true in order to trigger the elasticity action, specified by the policy. In a similar fashion, a condition can be expressed using Scheduled Expression, specifying how often the condition should evaluate to true. This is particularly useful for taking scaling decisions based on traffic patterns.

In Table 6, an Elasticity Action is described. Currently, the available actions refer to horizontal scaling, that is, replicating or removing resources. We should point out that a Resource is an abstract term and may refer either to a service or virtual machine flavor. The underline infrastructural resources are transparent to the Cloud Application Developer, as she only knows about the services of her application. The Unicorn platform is responsible about the infrastructural resources (e.g. virtual machines).

An Elasticity Action offers the ability to users and interested entities to specify a placement configuration for a service or an infrastructural resource in multiple clusters. A cluster can be located in the same zone or in another cloud provider, thus it supports multi-cloud scaling. Note that a placement configuration contains WarmupPeriod and CooldownPeriod production rules, which are important parameters for the efficiency of a scaling action and the overall performance of an application. The WarmupPeriod, is used to specify the time period that the metrics from newly instantiated resources are not considered by the respective auto-scaling handler. Similarly, the CooldownPeriod is used to give time to the system to provision/de-provision new resources and absorb any changes, so as to avoid false scaling alerts.

Table 4 — Elasticity Policy EBNF Definition

ElasticityPolicy	::=	ElasticityPolicyID “:” “When” ElasticityTriggerID “Perform” ElasticityActionID “With Priority” Priority
ElasticityPolicyID	::=	String
ElasticityTriggerID	::=	String
ElasticityActionID	::=	String
Priority	::=	PositiveInteger
PositiveInteger	::=	[1-9]([0-9])*

Table 5 — Elasticity Trigger EBNF Definition

ElasticityTrigger	::=	ElasticityTriggerID “:” ElasticityConditions
ElasticityTriggerID	::=	String
ElasticityConditions	::=	ElasticityCondition (“AND” ElasticityCondition)*
ElasticityCondition	::=	RuntimeInfoExpr | ScheduledExpr
RuntimeInfoExpr	::=	RuntimeInfo RelOp Number
RuntimeInfo	::=	InsightID | ResourceInfo “(” Members “)”
ResourceInfo	::=	“COUNT_RUNNING” | “COUNT_UNHEALTHY” | “HOURLY_COST”
Members	::=	Member (“,” Member)*
Member	::=	Resource |

		Resource “IN” Clusters
Clusters	::=	ClusterID (“,” ClusterID)*
ClusterID	::=	String
RelOp	::=	“LT” | “GT” | “EQ” | “NEQ” | “GTE” | “LTE”
Number	::=	[+-]? ([0-9]*[.])? [0-9]+
ScheduledExpr	::=	“REPEAT_EVERY” TimeExpression
TimeExpression	::=	Minute Hour DayOfMonth Month Weekday
Minute	::=	[0-5][0-9] (“,” [0-5][0-9])* | “*”
Hour	::=	([0-1][0-9] | [2][0-3]) (“,” ([0-1][0-9] | [2][0-3]))* | “*”
DayOfMonth	::=	([0-2][0-9] | [3][0-1]) (“,” ([0-2][0-9] | [3][0-1]))* | “*”
Month	::=	([1-9] | [1][0-2]) (“,” ([1-9] | [1][0-2]))* | “*”
Weekday	::=	[0-6] (“,” [0-6])* | “*”

Table 6 — Elasticity Action EBNF Definition

ElasticityAction	::=	ElasticityActionID “:” ResourceAction
ElasticityActionID	::=	String
ResourceAction	::=	ReplicationAction
ReplicationAction	::=	“ADD” PlacementAddConfigs | “REMOVE” PlacementConfigs
PlacementAddConfigs	::=	PlacementConfig “WITH” WarmupPeriod “,” CooldownPeriod (“,” PlacementConfig “WITH” WarmupPeriod “,” CooldownPeriod)*
PlacementConfig	::=	PositiveInteger Resource “IN” Cluster
PositiveInteger	::=	[1-9]([0-9])*
Resource	::=	String
Cluster	::=	String
WarmupPeriod	::=	PositiveInteger TimeUnit
TimeUnit	::=	“Milliseconds” | “Seconds” | “Minutes”| “Hours”
CooldownPeriod	::=	PositiveInteger TimeUnit
PlacementConfigs	::=	PlacementConfig (“,” PlacementConfig)*

Bibliography

[1] ISO 22093:2011, *Industrial automation systems and integration — Physical device control — Dimensional Measuring Interface Standard (DMIS)*