NebulOuS

A META OPERATING SYSTEM FOR BROKERING
HYPER-DISTRIBUTED APPLICATIONS ON
CLOUD COMPUTING CONTINUUMS

D3.1

INITIAL NEBULOUS BROKERAGE & RESOURCE MANAGEMENT

21/02/2024

Funded by
the European Union

Grant Agreement No.	101070516
Project Acronym/ Name	NebulOuS - A META OPERATING SYSTEM FOR BROKERING HYPER DISTRIBUTED APPLICATIONS ON CLOUD COMPUTINGCONTINUUMS
Topic	HORIZON-CL4-2021-DATA-01-05
Type of action	HORIZON-RIA
Service	CNECT/E/04
Duration	36 months (starting date 1 September 2022)
Deliverable title	Initial NebulOuS Brokerage & Resource Management
Deliverable number	D3.1
Deliverable version	2.0
Contractual date of delivery	31 December 2023
Actual date of delivery	21 February 2024
Nature of deliverable	Other
Dissemination level	Public
Work Package	WP3
Deliverable lead	University of Oslo
Author(s)	Geir Horn (Editor, University of Oslo),
Abstract	This deliverable is the textual enclosure to the resource management software produced for the first release of the NebulOuS platform. It intends to provide an overview and background of the implementation emphasising on the theoretical foundations and architecture of the software. It therefore serves as a necessary first introduction for a user of NebulOuS leaving the details and implementational details to the open-source code itself and the software wiki.
Keywords	Graphical User Interface, Resource Brokerage, Service ranking, Service Level Agreement, Optimisation, Workflow management

DISCLAIMER

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Directorate-General for Communications Networks, Content and Technology. Neither the European Union nor the granting authority can be held responsible for them.

COPYRIGHT

CONTRIBUTORS

Name	Organization
Geir Horn	University of Oslo
Marta Różańska	University of Oslo
Rudolf Schlatte	University of Oslo
Yiannis Verginadis	ICCS
Dimitris Apostolou	ICCS
Gregory Koronakos	ICCS
Fotis Paraskevopoulos	EXZ
Simeon Veloudis	SEERC
Ferran Diego Andilla	Telefónica

PEER REVIEWERS

Name	Organization
Fotis Paraskevopoulos	EXZ
Mario Reyes	Eurecat

REVISION HISTORY

Version	Date	Owner	Author(s)	Comments
1.0	04/02/2024	UiO	Geir Horn	Integration of contributions
1.1	10/02/2024	EXZ	Fotis Paraskevopoulos	Review
2.0	21/02/2024	UiO	Geir Horn	Finalisation

TABLE OF ABBREVIATIONS AND ACRONYMS

Abbreviation/Acronym	Open form
AHP	Analytic Hierarchy Process
AMPL	A Mathematical Programming Language
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ASET	Adaptive Scheduling of Edge Tasks
BQA	Brokerage Quality Assurance
CAMEL	Cloud Application Modelling and Execution Language
CC	Cloud Continuum
CLI	Command-Line Interface
CMS	Content Management System
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
DEA	Data Envelopment Analysis
DRL	Deep Reinforcement Learning
DSL	Domain Specific Language
GUI	Graphical User Interface
IO	Input-Output
IoT	Internet of Things
JSON	JavaScript Object Notation
MAPE-K	Monitor-Analyse-Plan-Execute with Knowledge
MCDM	Multiple-Criteria Decision Making

Abbreviation/Acronym	Open form
MDS	Metadata Schema
MOP	Multi-Objective programming
OM	Ontology Module
ORM	Object-Relational Mapping
OWL	Web Ontology Language
QoS	Quality of Service
RAM	Random-Access Memory
RBAC	Role-Based Access Control
REST	REpresentational State Transfer
RL	Reinforcement learning
SAL	Scheduling Abstraction Layer
SLA	Service Level Agreement
SLO	Service Level Objective
SMI	Service Measurement Index
SOA	Service Oriented Architecture
SPA	Single Page Application
UI	User Interface
UX	User Experience
YAML	Yet Another Markup Language [deprecated]

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

This deliverable is the textual enclosure to the resource management software produced for the first release of the NebulOuS platform. It intends to provide an overview and background of the implementation emphasising on the theoretical foundations and architecture of the software. It therefore serves as a necessary first introduction for a user of NebulOuS leaving the details and implementational details to the open-source code itself and the software wiki.

1 INTRODUCTION

This report is the textual enclosure to software deliverable *D3.1 Initial NebulOuS Brokerage & Resource Management*, category "other", and it introduces the software developed in NebulOuS Work Package 3, *Autonomous Deployments on Ad-hoc Cross-Clouds and Fog*. The aim is to give a high-level background to the implementation of the relevant research results, leaving the details about the code to the development wiki where the detailed interfaces and instructions for interacting with the NebulOuS code base are documented.

The main software modules of the NebulOuS platform and the related architecture were discussed in the deliverable *D2.1 Requirements and Conceptual Architecture of the NebulOuS Meta-OS*. The interaction diagram for the information flow among the platform components is shown in Figure 1, with the modules discussed in this deliverable highlighted. The structure of the deliverable follows the user application's path through the scope of WP3 starting with the user interface where the topological component model of the application is defined with the corresponding model for the metrics to monitor and the utility to be maximized by the running application. The resources available to the application is aggregated by the Clod/Fog Service broker, that provides a ranked list of resources applicable to use. The resources selected are validated against the operational constraints set for the application, leading to the definition of the Service Level Agreements (SLAs) to be enforced for the application deployment and reconfiguration.

The deployed application will be constantly monitored and optimised for the current execution context, and whenever a Service Level Objective (SLO) is violated, the Optimiser module will seek for a better application configuration, and the Optimiser module is documented as the last module covered by this deliverable. Workflow applications are special in the sense that they consist of a set of tasks with data dependent input constraints making the workflow application execution a scheduling problem. Workflow execution is therefore treated as a special type of application deployed on and execution infrastructure consisting of a scheduler and a set of workers on which the workflow application tasks will be executed, and where the NebulOuS platform is responsible for the provision of the optimised number of workers to be used by the workflow scheduler. The initial results on workflow scheduling closes this report.

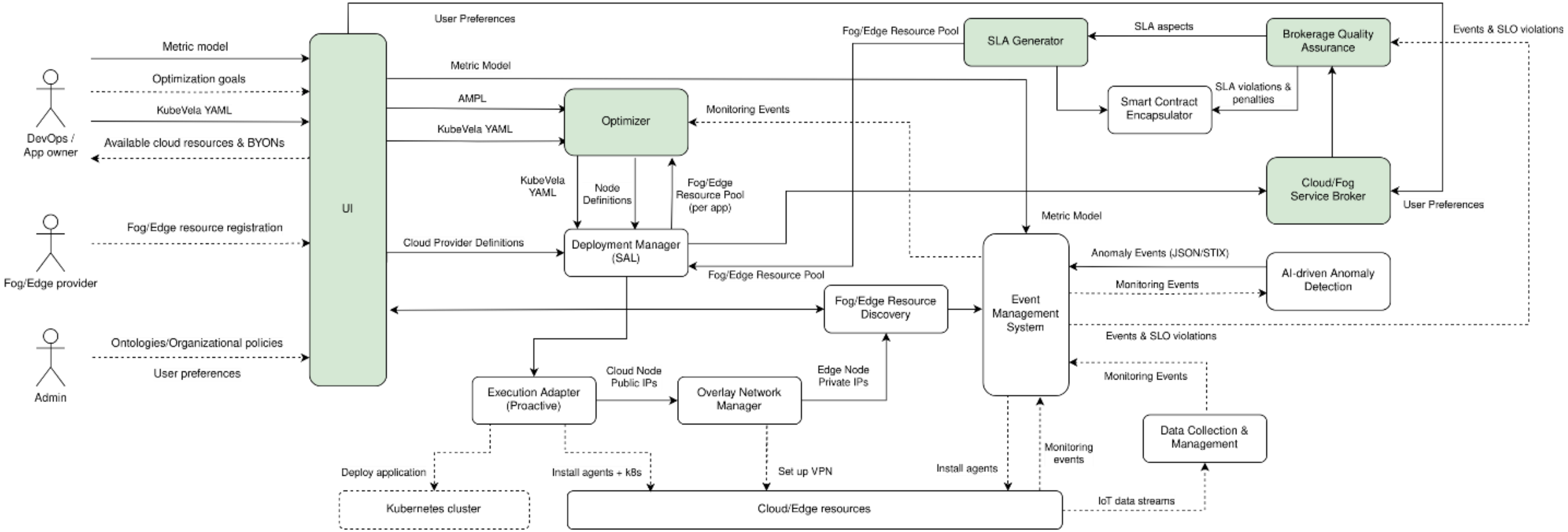


Figure 1: The component interaction diagram showing the components of the NebulOuS platform with the components developed in WP3 and introduced in this deliverable marked in green.

2 GRAPHICAL USER INTERFACE

To enhance the user's capacity for providing detailed information about the application being deployed, while also addressing necessary application and resource constraints, we have developed a Graphical User Interface (GUI) which simplifies the complexity that stems from deploying an application, describing application deployment properties, monitoring metrics and constraints, and resources available.

Central to the GUI are two primary components: the User Interface (UI) itself, which serves as the point of interaction for the users, and the application's controller (Controller), which contains the application logic. Together, these components create a cohesive and user-friendly environment for application deployment and management.

In the early stages of this prototype, we have focused on integrating key UI requirements as outlined in *D2.1 Requirements and Conceptual Architecture of the NebulOus Meta-OS*. These requirements encompass several critical areas:

- **Configuration:** We have empowered users to comprehensively define all aspects of their application using the UI, ensuring a seamless deployment process.
- **Monitoring:** The UI provides users with the capability to track their applications, offering insights into deployment status and operational metrics.
- **Security:** The UI segregates users based on their organizational affiliations through a Role-Based Access Control (RBAC) backend. Within this framework, we distinguish between two main roles: the organization administrator and the organization editor.
- **Resource Management:** Resource allocation and management are streamlined through an integration with the Scheduling Abstraction Layer (SAL). This integration allows organization administrators to register and manage resources effectively. An application editor is able to use these resources when defining the application.

In the subsequent sections, we will delve deeper into each of these areas. A detailed examination will be provided, along with a comprehensive walkthrough of the current version of the GUI. This will encompass both technical aspects and user interaction elements, illustrating how they collectively contribute to an effective and user-centric application deployment process.

2.1 PROTOTYPING AND IMPLEMENTATION ARCHITECTURE

In the initial stages of the design and implementation of the user interface (UI), we followed a structured and strategic approach. We used Figma[1] to create realistic wireframes and refine each aspect of the application's design from the outset. By using Figma, we were able to craft a detailed and accurate representation of the intended UI layout and functionality.

Building upon these wireframes, our approach involved crafting prototypes that served as functional representations of the UI. This method was particularly effective in communicating the envisioned UI to users and stakeholders. It enabled us to gather essential feedback early in the design process, ensuring that the UI was aligned with user needs and expectations. Once we received the necessary feedback, we continued in designing the architecture of the UI, and the tools and methods used for its implementation.

[1] https://www.figma.com

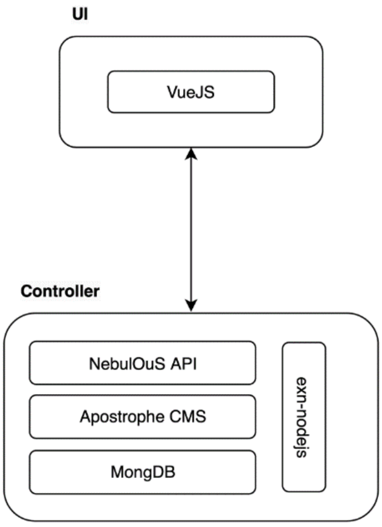


Figure 2: Figure 1 GUI Architecture

In keeping with the standards of modern application development, we opted to implement a Single Page Application (SPA). This decision was driven by the need for a responsive, dynamic, and user-friendly interface. The SPA was developed using the vue3.js[2] framework, chosen for its robustness, scalability, and suitability for creating reactive user interfaces.

Furthermore, for the controller, we opted for a powerful and efficient backend solution which could be use as the backbone of the component. We used the Apostrophe Content Management System (CMS[3]) as our backend framework, leveraging its advanced features such as Object-Relational Mapping (ORM) and Application Programming Interface (API) development tools. These features provided us with the flexibility and capability needed to build a robust and scalable UI.

Lastly, our communication with the NebulOuS platform was established using asynchronous communication, utilizing the **exn-nodejs-library**. This approach ensured seamless and efficient data exchange, crucial for maintaining the responsiveness and reliability of the UI.

2.2 USER INTERFACE WALKTHROUGH

In this section we will describe and go over the core features of the UI prototype.

2.2.1 User Management

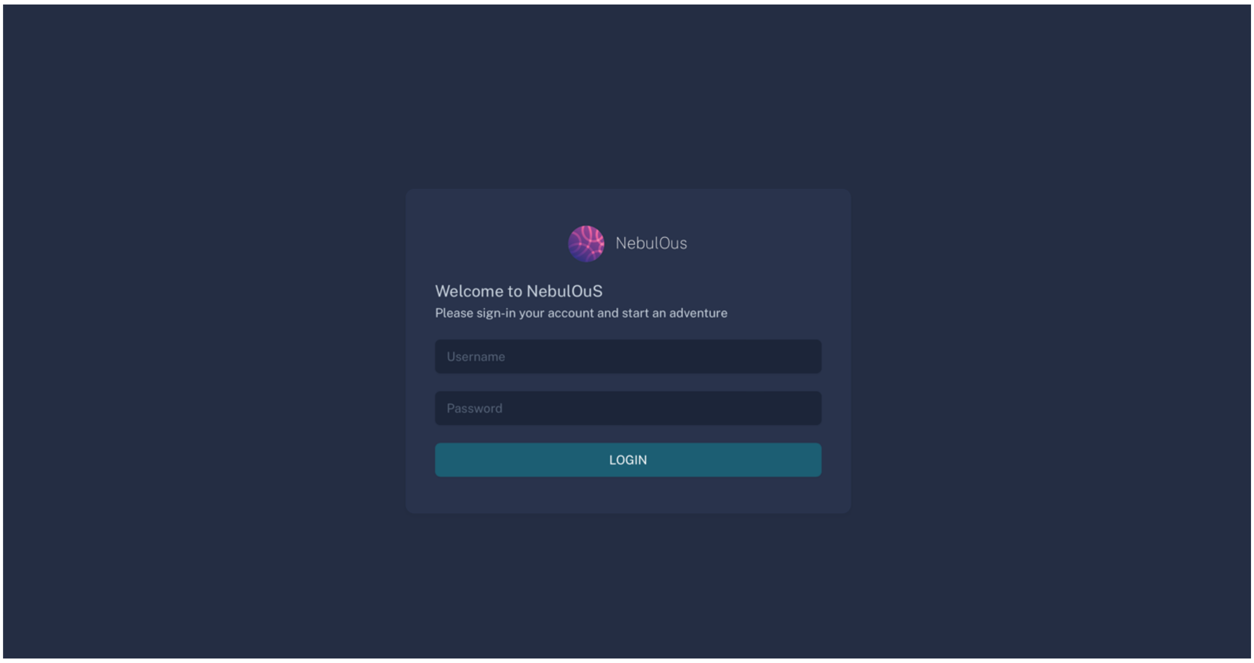
There are three essential roles regarding the users that interact with the UI.

1. **The NebulOuS administrator (superadmin):** Has access to the ApostropheCMS backend and is able to globally manage all data aspects of the UI.
2. **The organization administrator (admin):** This user is the bootstrapped by the *superadmin*, and belongs to an organization. The administrator can manage users (editors) within their organization, as well as manage Resources (SAL) which are available during the application creation process.
3. **The organization user (editor):** This user is created by the *admin*, they belong to a single organization, they are allowed to manage applications, as well as modify their own profile through the UI.

Once the *admin* user is onboarded, the admin can login into the UI and create the users of the organizations.

[2] https://vuejs.org

[3] https://v3.docs.apostrophecms.org

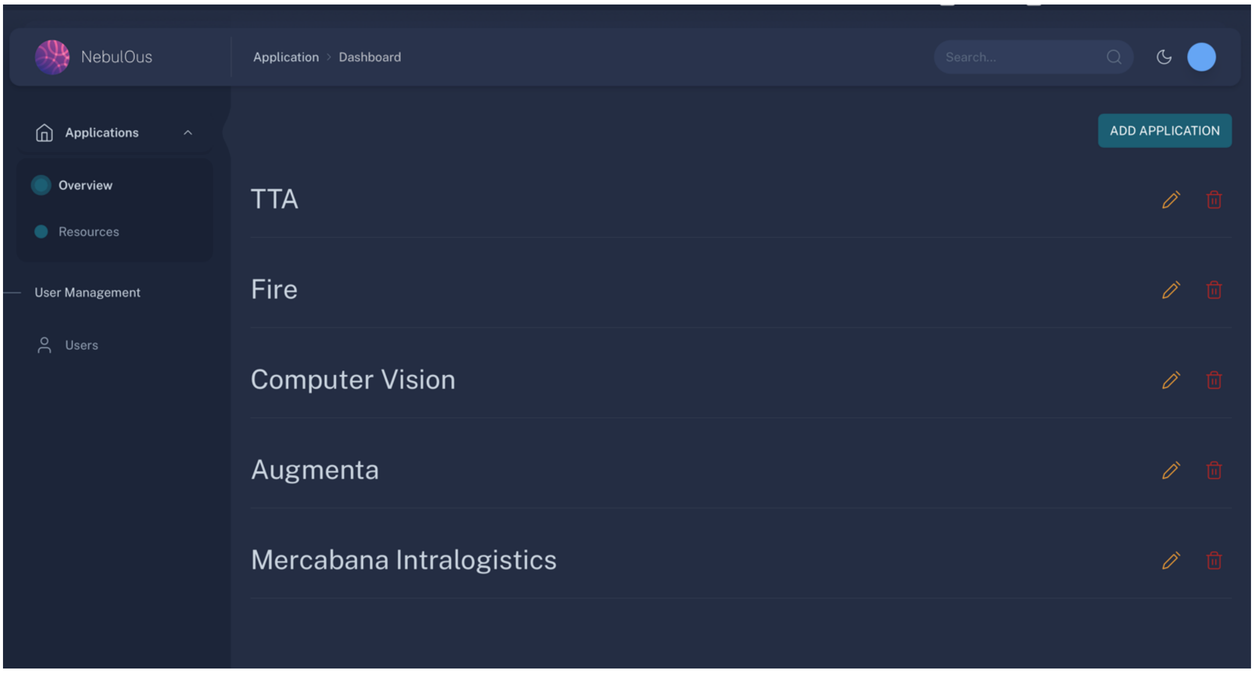


Welcome to NebulOuS

Please sign-in your account and start an adventure

Figure 3: UI Login

Once logged in the user is presented with the application already created within the same organization. From this screen the user can review the deployment status of applications, edit or create a new application.



TTA

Fire

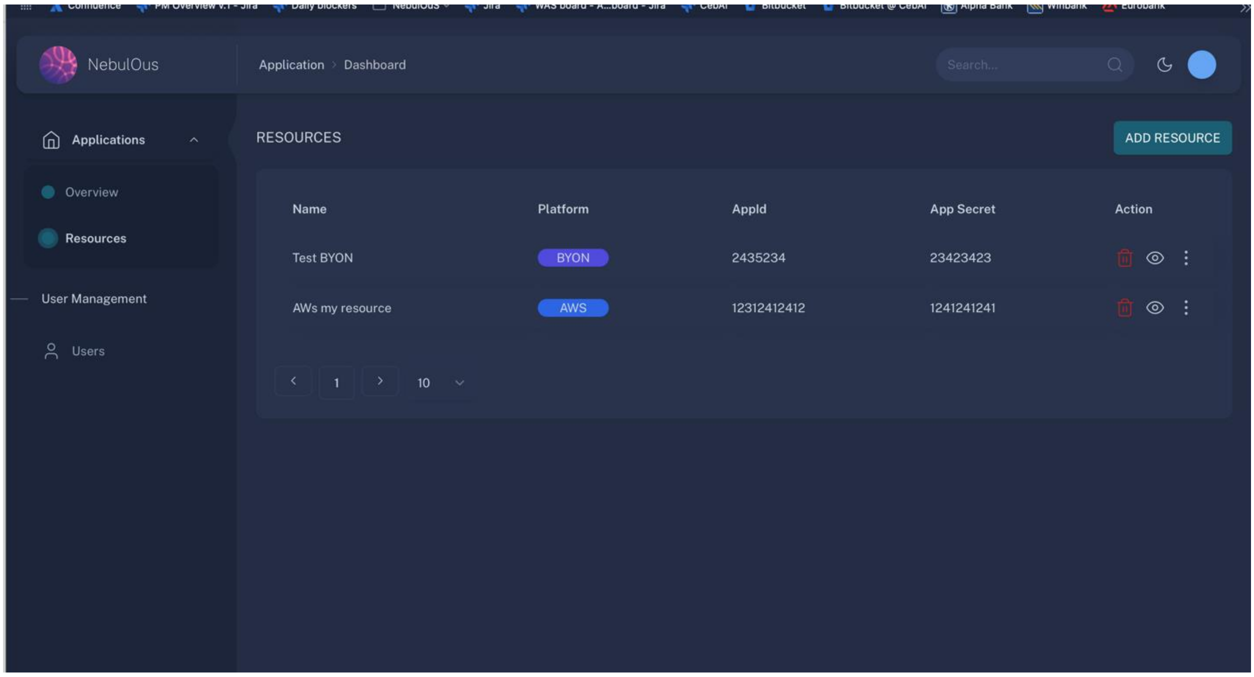
Computer Vision

Augmenta

Mercabana Intralogistics

Figure 4: Application List

Before creating an application, an *admin* needs to register the available resources for their organization, which will be used during application deployment.



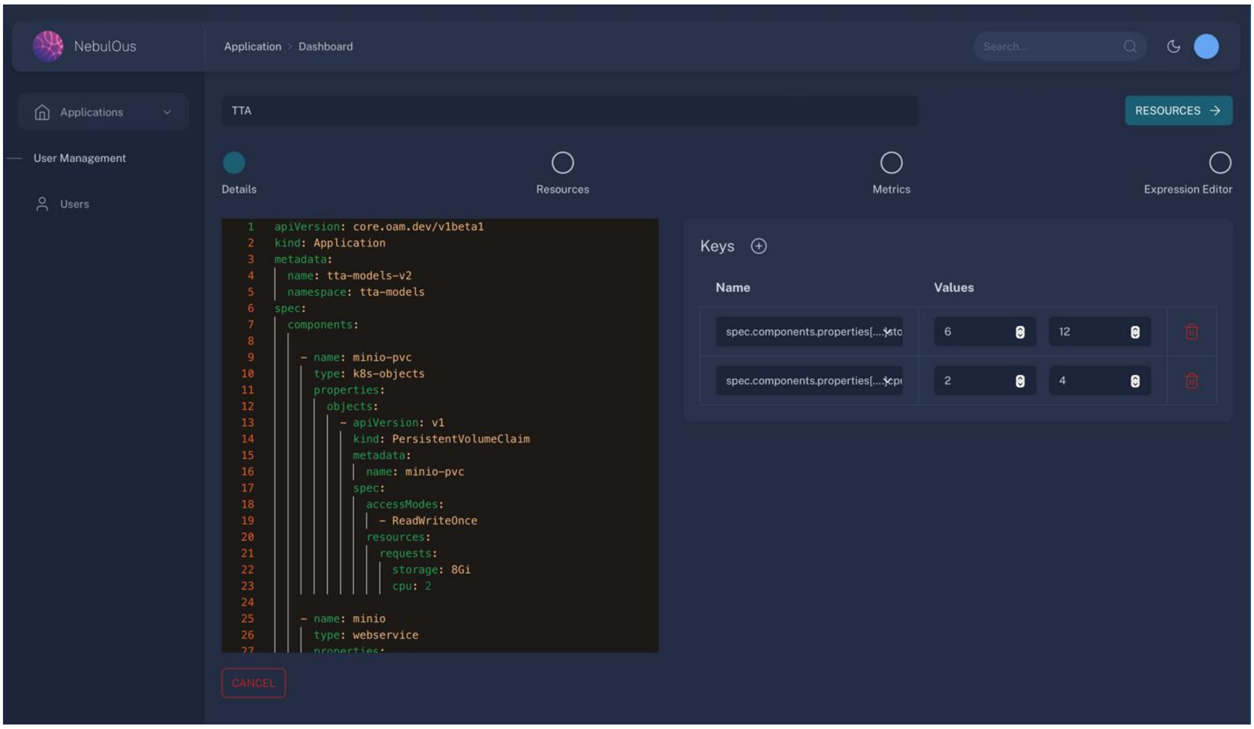
Name	Platform	AppId	App Secret	Action
Test BYON	BYON	2435234	23423423	
AWs my resource	AWS	12312412412	1241241241	

Figure 5: Resources Manager

Once the available resources have been set a user can then create an application to be deployed. In order to assist the user during the provision of application data, we opted for a step-wizard like interface, validating the information provided by the user at each step, and cross-checking that variables, metrics, and functions are consistent at the data level.

Step 1 - Application Description

Here we provide a deployable KubeVela file and provide the user with the ability to define which sections of the KubeVela file, can be used and altered during the optimization process.



```yaml
apiVersion: core.oam.dev/v1beta1
kind: Application
metadata:
  name: tta-models-v2
  namespace: tta-models
spec:
  components:

    - name: minio-pvc
      type: k8s-objects
      properties:
        objects:
          - apiVersion: v1
            kind: PersistentVolumeClaim
            metadata:
              name: minio-pvc
            spec:
              accessModes:
                - ReadWriteOnce
              resources:
                requests:
                  storage: 8Gi
                  cpu: 2

    - name: minio
      type: webservice
      properties:
```

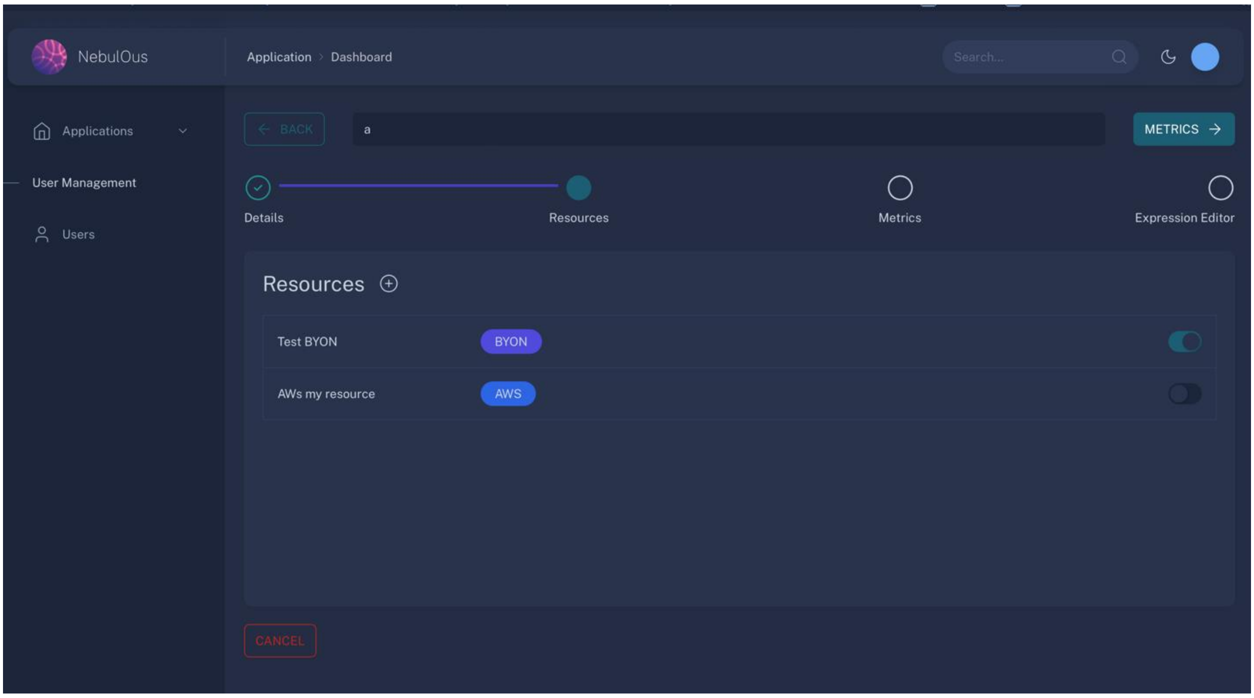
Keys

Name	Values	
spec.components.properties[....]sto	6	12
spec.components.properties[....]cpu	2	4

Figure 6: The KubeVela application model view

Step 2 – Resource Selection

Here the user can determine which resources, that of the available resources for the organization can be used during the deployment and monitoring of the application.

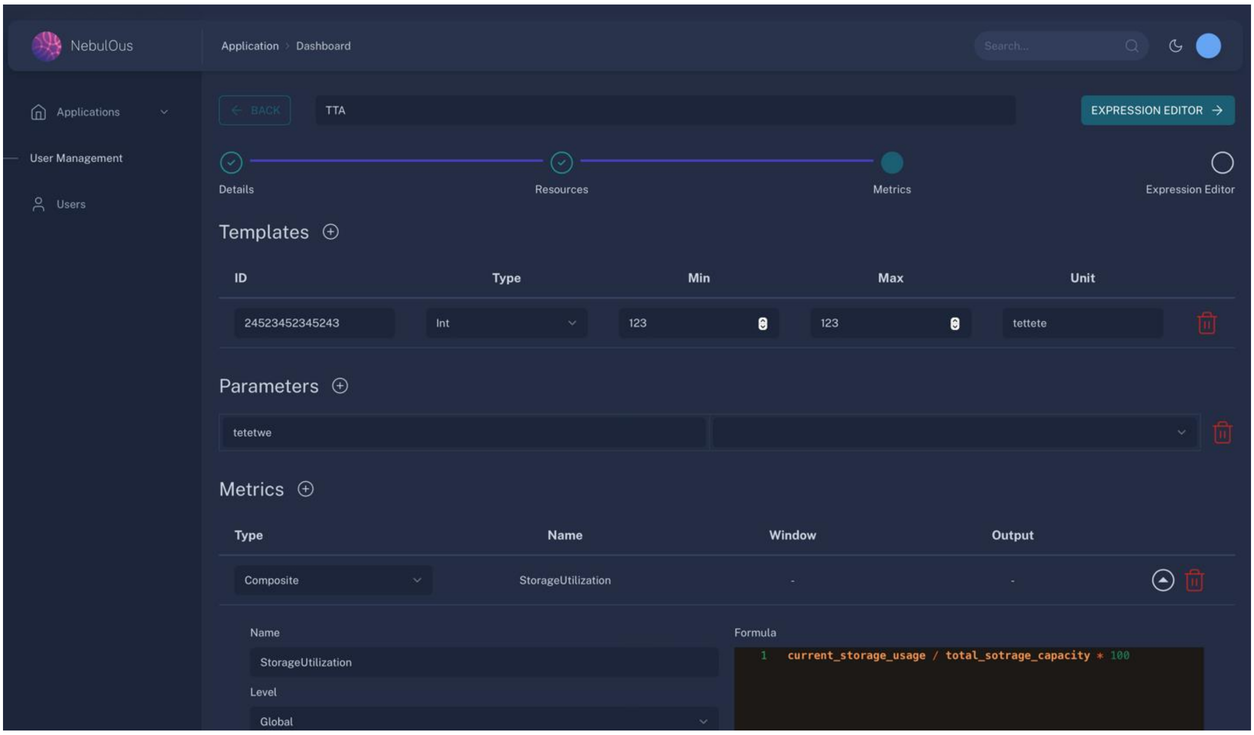


Test BYON	BYON	● on
AWs my resource	AWS	○ off

Figure 7: Resource Selection

Step 3 – Metric Model & SLO Definition

During this step we provide the user with the necessary UI functionality to fully describe the metrics that are available for their application, and how these will be monitored. Furthermore, we allow the user to define SLOs by specifying constrains on these metrics. The input during this process will be used to generate the Metric Model Domain Specific Language (DSL) by the controller of the UI.



TTA

Templates ⊕

ID	Type	Min	Max	Unit
24523452345243	Int	123	123	tettete

Parameters ⊕

tetetwe

Metrics ⊕

Type	Name	Window	Output
Composite	StorageUtilization	-	-

Name: StorageUtilization

Level: Global

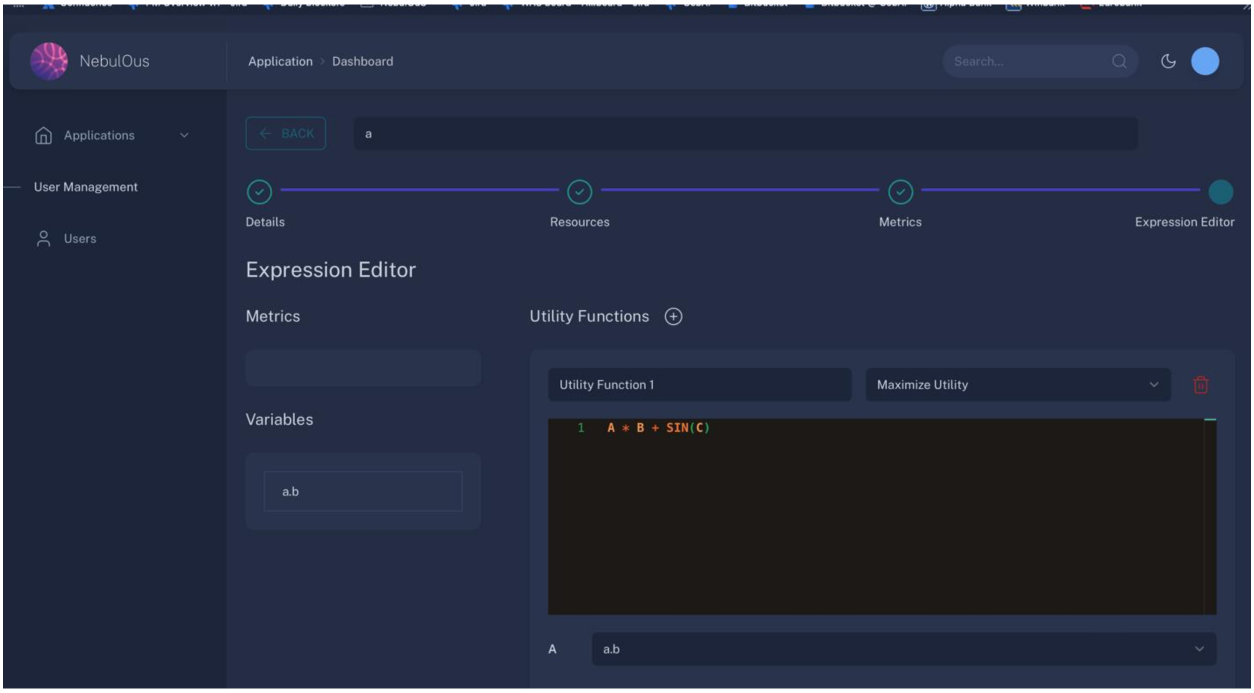
Formula

```
current_storage_usage / total_sotrage_capacity * 100
```

Figure 8: The step defining the metric model and the problem constraints.

Step 4 – Expression Editor

This is the last step of the application definition process, where the user is able to input utility functions using math syntax. The UI extracts the constants used in the formula and allows the user to declaratively specify the metrics and variables that the constant refers to. The value can be a KubeVela path defined in **Step 1,** or metric defined in **Step 3,** and finally any other pre-defined Utility Function, allowing for extensible cascading.



a

Utility Function 1 — Maximize Utility

```
A * B + SIN(C)
```

a.b

A: a.b

Figure 9: Expression Editor - defining utility functions

At each step of the process the UI through the controller, performs validation on the data input by the user, to ensure that all information is coherent, and be used during the deployment of the application. The user can save the application definition and perform the deployment of the application through the application list. Finally, once the application has been properly defined and saved, the controller is responsible for a sequence of event described in the next section.

2.3 CONTROLLER

As aforementioned the controller component contains the logic which complemented the User Interface. The UI communicates with the controller via a RESTful API, and the functionality in this prototype includes.

- **Data Validation:** The controller provides the UI with a validation layers. Here we parse the KubeVela file, along with all mathematical expressions defined through the application, and perform variable, metric, and naming validation, in order to ensure that once the information for the application is provided this can be used by the rest of the components.
- **DSL Generation:** Once the application is by the user, the controller layer communicates via asynchronous messaging to the rest of the components that a new application has been defined, along with two DSLs, one is the metric model in YAML format and the second is a full application description in JSON format.
- **API Interface** The component also allows external systems to perform actions in the same manner that a user would be able to do through the UI. The is to allow for example automated flows, or systemic calls related to the application.

2.4 FUTURE STEPS

Our efforts so far have laid a solid foundation, but several key developments are planned to further enhance the system's capabilities and user experience. In the next version of the UI we will implement push-based real-time monitoring, using a time-series database and exchanging application data in the form of asynchronous messages. Furthermore, we will introduce and implement ontology based (SLO) validation, in order to ensure that user defined SLO constraints are applicable and possible within the context of the application description.

We also plan to extend the API Interface to cover all system functionalities. And this will form the basis of a Command-Line Interface (CLI) compatible with various *nix systems so that users can interact with the NebulOuS through the command line. This will cater to a broader range of users, especially those who prefer or require command-line access, thereby enhancing the system's accessibility.

Finally, the User Experience (UX) of the UI will undergo continuous refinement, particularly in response to feedback from initial user trials. This iterative process will ensure that the UI is not only functional but also intuitive and user-friendly.

3 CLOUD CONTINUUM BROKERAGE

3.1 PURPOSE

As the complexity of the Cloud Continuum (CC) increases, the role of brokers in the cloud continuum ecosystems becomes increasingly important. With the increase of cloud technologies adoption, the number of services offered in the cloud market as well as the availability of cloud continuum computing resources also raises. Thus, the evaluation of the available cloud continuum computing resources can be a cumbersome task for the user due to the plethora of the offered services in the cloud market, the heterogeneity of edge and fog devices, and the lack of standard mechanisms that allow their comparison against user requirements. In that respect, there is an increasing need for user guidance during the computing resource selection process. Cloud brokers that mediate between the user and the cloud continuum ecosystem assist the user in filtering out not performing or not suitable resources and selecting the most appropriate resource.

The multidimensional nature of cloud services involves several factors for their evaluation, such as performance, availability, security, etc. A performance evaluation technique that is adequate to handle multiple factors and aggregate them to a score for each assessed entity is the Data Envelopment Analysis (DEA) [1], which is one of the foundational techniques in Operations Research [2]. In the context of Nebulous, we integrate DEA with Multiple-Criteria Decision Making (MCDM) methods to evaluate the CC nodes. In particular, we utilize DEA and Multi-Objective Programming (MOP) to derive the ranking of the fog nodes. Also, we incorporate the users' preferences with respect to the relative importance of the evaluation criteria. We introduce these value judgments into the optimization models as weight restrictions [3]. Beyond the incorporation of the relative importance of the criteria, ordinal information about the fog nodes can be also incorporated into the assessment. In the context of the proposed approach that integrates DEA with MOP, this is implemented by including a categorization of some of the alternative fog nodes into the evaluation models. The categorization is obtained from the users.

3.2 INTERNAL ARCHITECTURE & INTERFACES

The architecture of the Broker is shown in Figure 10. The CC nodes fetcher sub-component retrieves available cloud continuum nodes from the NebulOuS Resource Discovery component. The Criteria selection sub-component is responsible for managing, displaying to the user and allowing the selection of the criteria according to which the CC nodes will be evaluated. The Preference elicitation sub-component allows the user to define preferences; this is an optional step. Finally, the Solver evaluates the available CC nodes and generates a ranked list of CC nodes, taking also into account any user preferences.

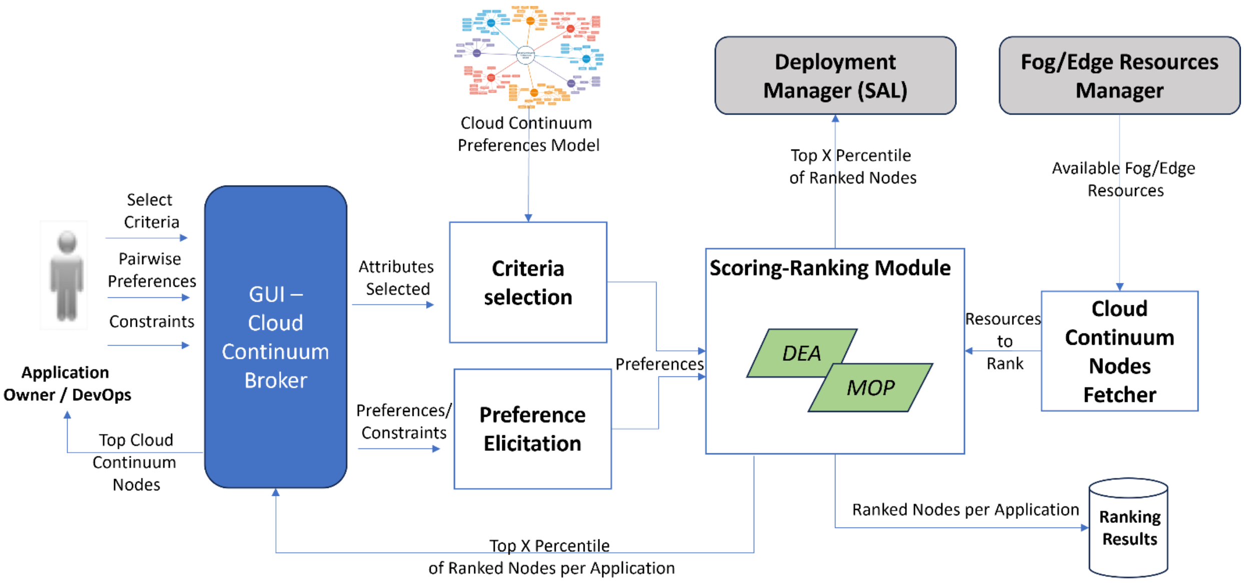

Cloud Continuum Preferences Model
Deployment Manager (SAL)
Fog/Edge Resources Manager
Top X Percentile of Ranked Nodes
Available Fog/Edge Resources
Select Criteria
Pairwise Preferences
Constraints
Application Owner / DevOps
GUI – Cloud Continuum Broker
Attributes Selected
Criteria selection
Scoring-Ranking Module
DEA
MOP
Preferences
Resources to Rank
Cloud Continuum Nodes Fetcher
Preferences/ Constraints
Preference Elicitation
Top Cloud Continuum Nodes
Ranked Nodes per Application
Ranking Results
Top X Percentile of Ranked Nodes per Application

Figure 10: NebulOus CC Brokerage Architecture

CC Brokerage makes use of interfaces for persisting the selected criteria (Table 1), user preferences (Table 2), and generated output (Table 3).

Table 1. Interface Selected Criteria

JSON
{ "fog_node_candidates_criteria": [{ "name": "Memory Speed", "type": "Quantitative", "measure": "MHz" }, { "name": "Cost", "type": "Quantitative", "measure": "Euro" }, { "name": "Security", "type": "Qualitative", "measure": "Ordinal Scale 1-3" }] }

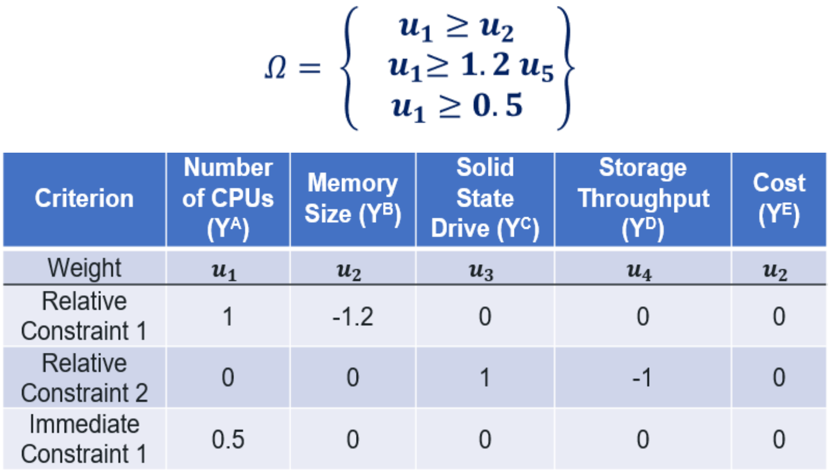
```
{
"fog_node_candidates_criteria":
 [
  {
   "name": "Memory Speed",
   "type": "Quantitative",
   "measure": "MHz"
  },
  {
   "name": "Cost",
   "type": "Quantitative",
   "measure": "Euro"
  },
  {
   "name": "Security",
   "type": "Qualitative",
   "measure": "Ordinal Scale 1-3"
  }
 ]
}
```

The attribute *type* in JSON provided in Table 2 reflects the preference of a user over a pair of criteria, e.g., “ge” denotes greater than or equal to. In addition, the attribute *constraint* includes the information about the magnitude of preference between two criteria and the corresponding constraint that will be used in the optimization models. Except from relative constraints between two criteria, immediate ones can be given to bound (from above or below) the weight of a criterion in the optimization process.

Table 2. User Preferences

JSON	Example

```
{
 "fog_node_user_preferences":
  [
   {
    "type": "ge",
    "constraint": [1, -1, 0, 0, 0]
   },
   {
    "type": "ge",
    "constraint": [1, 0, 0, 0, -1.2]
   },
   {
    "type": "ge",
    "constraint": [0.5, 0, 0, 0, 0]
   }
  ]
}
```



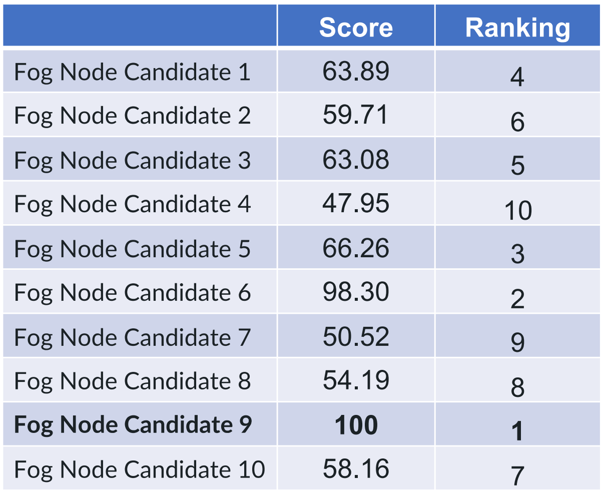
$$\Omega = \left\{ \begin{array}{c} u_1 \geq u_2 \\ u_1 \geq 1.2\, u_5 \\ u_1 \geq 0.5 \end{array} \right\}$$

Criterion	Number of CPUs (Y^A)	Memory Size (Y^B)	Solid State Drive (Y^C)	Storage Throughput (Y^D)	Cost (Y^E)
Weight	u_1	u_2	u_3	u_4	u_2
Relative Constraint 1	1	-1.2	0	0	0
Relative Constraint 2	0	0	1	-1	0
Immediate Constraint 1	0.5	0	0	0	0

Table 3. Interface Output example

JSON	Example

```
{
"fog_node_candidates":
  [
   {
    "name": "Fog Node Candidate 1",
    "score": 63.89,
    "ranking": 4
   },
....
...
{
    "name":"Fog Node Candidate 10",
    "score":58.16,
    "ranking":7
   }
  ]
}
```



	Score	Ranking
Fog Node Candidate 1	63.89	4
Fog Node Candidate 2	59.71	6
Fog Node Candidate 3	63.08	5
Fog Node Candidate 4	47.95	10
Fog Node Candidate 5	66.26	3
Fog Node Candidate 6	98.30	2
Fog Node Candidate 7	50.52	9
Fog Node Candidate 8	54.19	8
Fog Node Candidate 9	**100**	**1**
Fog Node Candidate 10	58.16	7

3.3 IMPLEMENTATION

For the NebulOuS CC Brokerage to be capable of comparing different CC resources and providing a ranked list of offerings, an appropriate model for describing their comparable characteristics is imperative.

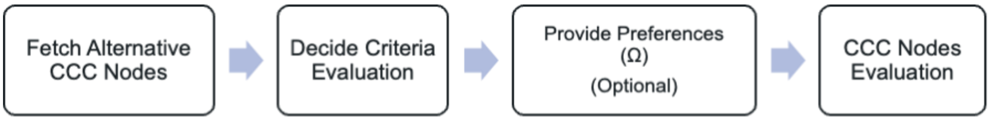

Fetch Alternative CCC Nodes
Decide Criteria Evaluation
Provide Preferences (Ω) (Optional)
CCC Nodes Evaluation

Figure 11: Methodology

We employ the methodology shown in Figure 11 to implement CC Brokerage. The first step is to fetch all available CC nodes (resources). Next, the user can select among NebulOus CC Attribute model the ones which are relevant to be used as criteria for the evaluation of available nodes. Finally, the user can (optionally) declare his or her preferences, and the component generates a ranked list of available CC nodes.

The NebulOuS Cloud/Fog Service Broker to be capable of comparing different cloud continuum resources and providing a ranked list of offerings, an appropriate model for describing their comparable characteristics is imperative. The ranked list of available Fog and Edge resources will be used by NebulOuS for creating ad-hoc cloud continuums dedicated to host application components instances. The model should encapsulate all the necessary user preference indicators that will allow for comparisons between cloud continuum resources. The model is presented in Figure 3 and essentially involves the reuse, extension and proper adjustment of mainly the Service Measurement Index (SMI) [4], and the Broker@Cloud preferences model [5], which have been introduced for capturing preferences over cloud services. Both involve a hierarchical framework that divide the measurement space into 7 and top-level categories, respectively, that are further refined by 3 or more attributes as seen below. The NebulOuS preferences model is depicted using a mind map notation to provide a good overview of the attributes along with their hierarchy involved.

All the attributes are analysed in the Appendix, providing indications on the updates/adjustments provided over SMI and Broker@Cloud models.

We employ DEA to obtain a performance score for each fog node based on the aggregation of the criteria. DEA is a data driven technique for the performance evaluation of a set of comparable entities with several attributes (criteria) assumed as inputs and outputs, i.e., each entity converts multiple inputs to multiple outputs. The composite score FN_Score_j for the specific fog node j (*j=1,...,n*) derives as the weighted sum $FN\_Score_j = uY_j$, where $Y_j = \left(Y_{j1}, Y_{j2}, \dots, Y_{jm}\right)^T$ denotes the vector of the values of the m criteria and $u = (u_1, u_2, \dots, u_m)$ denotes the vector of the variables used as weights.

$$\begin{aligned} & \max FN\_Score_{j_0} = uY_{j_0} \\ & subject\ to \\ & uY_j \leq 1, \ \ j = 1, \dots, n \\ & u \geq 0 \end{aligned} \tag{1}$$

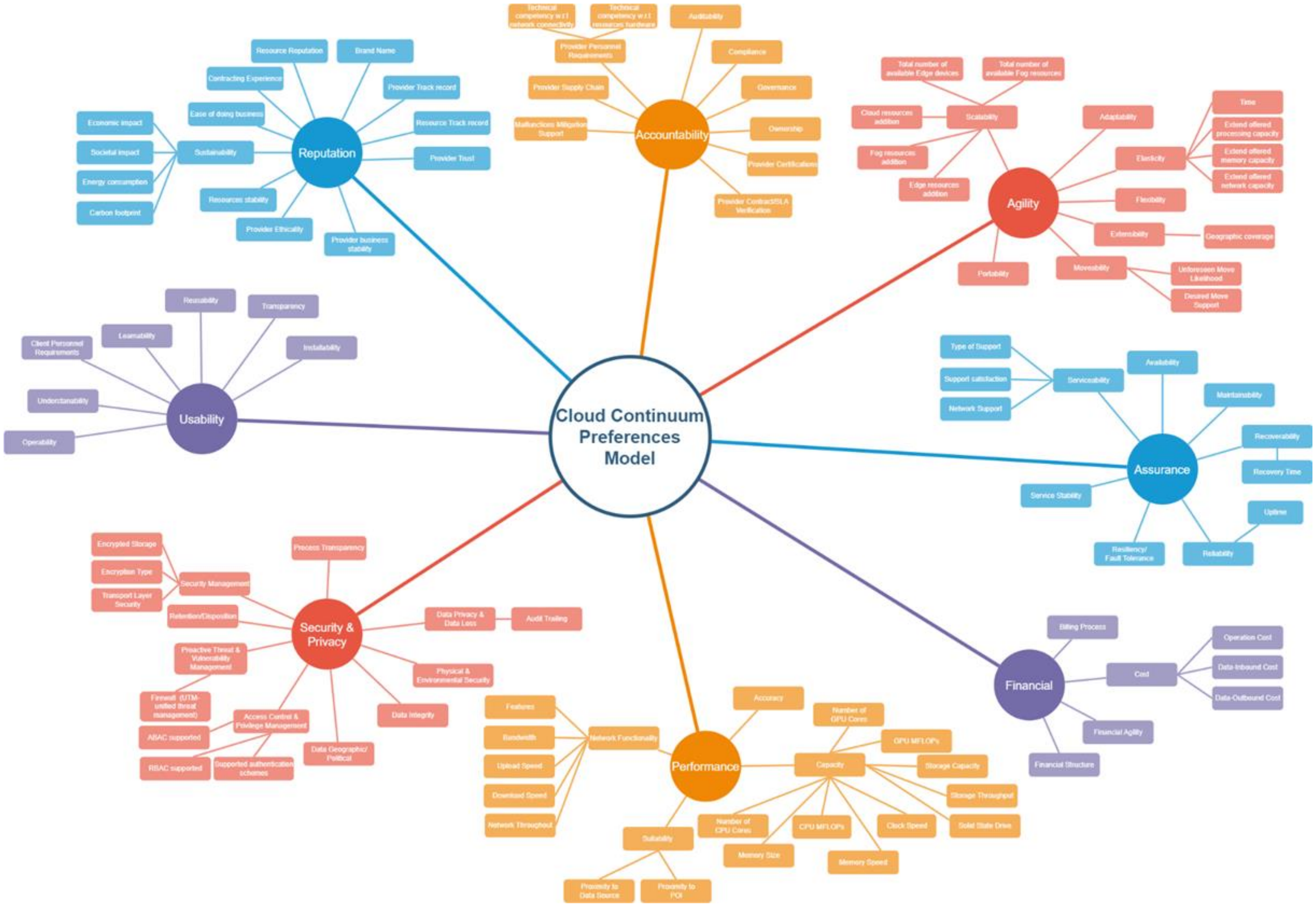


Figure 12: NebulOus CC Attribute Model

Funded by the European Union

Model (1) is a linear program that is solved for one fog node at a time to maximize its score. We note that for some criteria, denoted by set *UC*, lower values are preferred (the less the better) as higher values correspond to worser performance. For instance, for the criterion *Cost* of each fog node. The values ($\hat{Y}$) of such criteria are inverted to exhibit positive contribution [6].

$$Y_k = 1/\hat{Y}_q\,, \quad \mathrm{q} \in \mathrm{UC};\, |\mathrm{UC}| \leq \mathrm{m} \tag{2}$$

Apart from deriving a score for each fog node, a ranking of them is required to identify the best ones [7]. As the conventional DEA models, CCR [8] and BCC [9], are solved for each evaluated entity, they yield different weighting schemes for each fog node that allow the maximization of their score. Therefore, we integrate MOP into our approach to obtain a common weighting scheme to aggregate the criteria of the fog nodes and determine their ranking. MOP and DEA are similar in structure, the relationships between them are explored [10].

As the score of each fog node is calculated by model (1) separately from the others, the optimal multipliers u^* vary from plan to plan. The different fog node-specific weighting schemes derived by model (1) allow each fog node to achieve the highest possible score (FN^*_{score}). A common basis for comparisons and ranking can be established by finding a common set of multipliers u that will be used to obtain the score of each fog node. For this purpose, we formulate the following MOP model where the performance of each fog node ($FN\_Score_j = uY_j$) is treated as a distinct objective.

$$\begin{aligned} &\max FN_{Score_1} = uY_1 \\ &\quad \vdots \\ &\max FN_{Score_n} = uY_n \\ &subject\ to \\ &uY_j \leq \mathrm{FN}^*_{\mathrm{score}_j}\,, \quad j = 1, \dots, n \\ &u \geq 0 \end{aligned} \tag{3}$$

Several methods have been developed for solving multi-objective programming problems [11]. We use the scalarization method to convert MOP (3) to a single objective program. In particular, we utilize the *method of the global criterion* [12], which is a no-preference method, i.e., no priority is assigned to the objectives. However, a variant of this method that incorporates preference information from the users can be straightforwardly applied alternatively. In global criterion method, the distance between some reference points and the feasible objective region is minimized. We select the reference point $RP = (FN^*_{\mathrm{score}_1}, \dots, FN^*_{\mathrm{score}_n})$ that contains the highest possible score attained by each fog node using model (1). The distance between the reference point and the feasible objective region can be measured by employing different metrics as the following L_p problem exhibits.

$$\begin{aligned} &min \left(\sum_{j=1}^{n} \left| FN^*_{\mathrm{score}_j} - uY_j \right|^p \right)^{1/p} \\ &subject\ to \\ &uY_j \leq \mathrm{FN}^*_{\mathrm{score}_j}\,, \quad j = 1, \dots, n \\ &u \geq 0 \end{aligned} \tag{4}$$

We scalarize MOP (3) via the method of the global criterion by employing the L_1 metric, i.e., $p=1$ in model (4).

$$
\begin{aligned}
& min \sum_{j=1}^{n} (\mathrm{FN}^{*}_{\mathrm{score}_j} - uY_j) \\
& subject\ to \qquad\qquad\qquad\qquad (5) \\
& uY_j \leq \mathrm{FN}^{*}_{\mathrm{score}_j},\ \ j = 1, \dots, n \\
& u \geq 0
\end{aligned}
$$

The single objective model (5), also known as the *min-sum* method, is solved only once and simultaneously minimizes the sum of the deviations (L_1 metric) for the fog nodes between the performance that they can achieve using the common multipliers and the highest one ($\mathrm{FN}^{*}_{\mathrm{score}}$). In other words, the aim of model (5) is to maximize as much as possible the scores of all fog nodes under a common weighting scheme. Model (5) is straightforwardly transformed to model (6) by introducing the deviation variables ($d_j = \mathrm{FN}^{*}_{\mathrm{score}_j} - uY_j$) at the constraints and replacing the corresponding terms in the objective function.

$$
\begin{aligned}
& min \sum_{j=1}^{n} d_j \\
& subject\ to \qquad\qquad\qquad\qquad (6) \\
& uY_j + d_j = FN^{*}_{\mathrm{score}_j},\ \ j = 1, \dots, n \\
& u \geq 0, d_j \geq 0
\end{aligned}
$$

Models (5) and (6) are equivalent and provide higher discrimination than model (1) regarding the performance of the evaluated fog nodes. Also, these models allow for ranking since all fog nodes collectively and equally participate to the generation of the optimal set of weights used for the calculation of their scores. The optimal solution of models (5) and (6) is Pareto optimal to MOP (3).

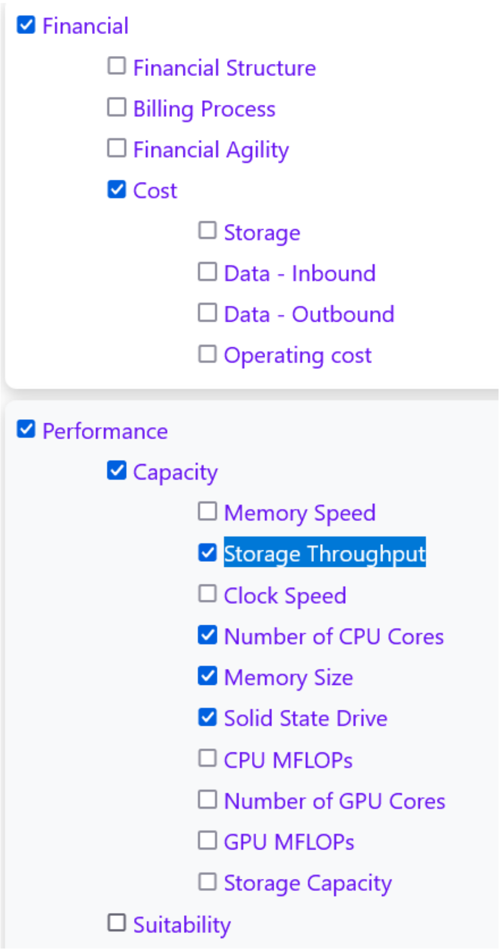
The priorities of the users over the criteria can be also incorporated into the proposed evaluation models by translating them into weight restrictions [13]. This can be implemented either by including the users' explicit preference over two criteria, e.g., $u_1 \geq 2u_2$ or by utilizing Analytic Hierarchy Process (AHP) to elicit the users' preferences [14]. In particular, the users' priorities are used to obtain assurance region constraints that restrain the weights (u) of the selected criteria.

The explicit expression of user's priorities over a pair of criteria results in lower and/or upper bounds (L_{ij}, U_{ij}) that limit the weight assigned to each criterion. Specifically, for every pair of criteria (i, j) the ratio of their weights (u_i/u_j) is bounded as follows.

$$L_{ij} \leq u_i/u_j \leq U_{ij} \qquad (7)$$

We denote the whole set of weight restrictions with Ω, $u \in \Omega$. The set Ω generally denotes restrictions imposed on the weights that limit the freedom of the evaluated fog nodes in selecting the optimal weights to maximize their scores [15].

Our approach is illustrated by generating a data set with ten fog nodes using five criteria from the available ones as depicted in Figure 13.



- ☑ Financial
  - □ Financial Structure
  - □ Billing Process
  - □ Financial Agility
  - ☑ Cost
    - □ Storage
    - □ Data - Inbound
    - □ Data - Outbound
    - □ Operating cost
- ☑ Performance
  - ☑ Capacity
    - □ Memory Speed
    - ☑ Storage Throughput
    - □ Clock Speed
    - ☑ Number of CPU Cores
    - ☑ Memory Size
    - ☑ Solid State Drive
    - □ CPU MFLOPs
    - □ Number of GPU Cores
    - □ GPU MFLOPs
    - □ Storage Capacity
  - □ Suitability

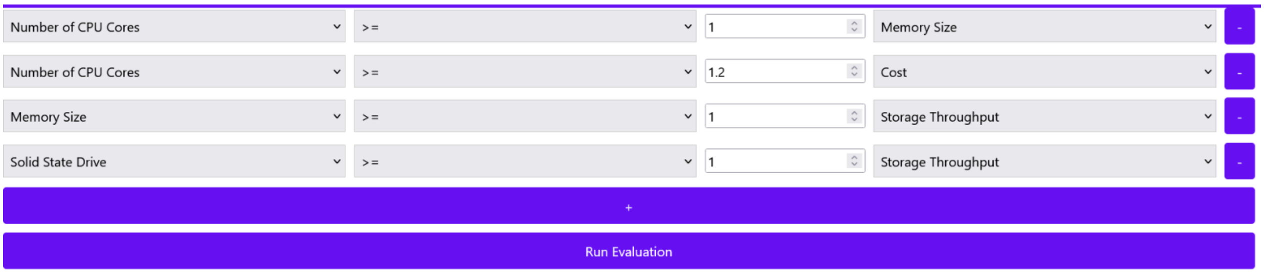
Figure 13: Criteria Selection via Broker component

Table 4 exhibits the profile of the ten fog nodes.

Table 4. Data for ten fog nodes and five criteria

	Number of CPUs (Y^A)	Memory Size (Y^B)	Solid State Drive (Y^C)	Storage Throughput (Y^D)	Cost (Y^E)
Fog Node Candidate 1	2	4	150	1500	419
Fog Node Candidate 2	8	6	200	4000	553
Fog Node Candidate 3	16	32	300	5600	1152
Fog Node Candidate 4	4	6	500	3300	673
Fog Node Candidate 5	32	16	600	6000	1853
Fog Node Candidate 6	44	16	2100	4800	2985
Fog Node Candidate 7	4	8	1500	9500	972
Fog Node Candidate 8	12	8	2500	11000	1756
Fog Node Candidate 9	48	32	5000	5500	3738
Fog Node Candidate 10	12	16	3000	12000	2348

In the context of the illustrative example the user conducting the assessment expressed the following priorities over the criteria: $\Omega = \{u_1 \geq u_2,\ \ u_1 \geq 1.2\, u_5\ , u_2 \geq u_4, u_3 \geq u_4\ \}$. A screenshot of these restrictions formed by the Broker component is shown in Figure 14.



Number of CPU Cores	>=	1	Memory Size
Number of CPU Cores	>=	1.2	Cost
Memory Size	>=	1	Storage Throughput
Solid State Drive	>=	1	Storage Throughput

Figure 14: Preference Elicitation via Broker component

We incorporate into the evaluation models (1) and (6) the weight restrictions Ω to carry out the assessment of the 10 fog nodes. We notice that prior to applying the models, the criterion *Cost* is converted to exhibit positive contribution using formula (2). The highest possible score attained by each fog node using model (1) with Ω is reported in the second column of Table 5. The scores and the ranking derived from model (6) with Ω are reported in columns 3-4 of Table 5. These are calculated using a common weighting scheme for all fog nodes.

Table 5. Performance scores and Ranking of the ten fog nodes

	Scores from model (1) with Ω	Scores from model (6) with Ω	Ranking
Fog Node Candidate 1	80.01	79.95	3
Fog Node Candidate 2	72.98	72.97	5
Fog Node Candidate 3	78.16	58.20	6
Fog Node Candidate 4	55.06	55.02	7
Fog Node Candidate 5	78.20	78.18	4
Fog Node Candidate 6	94.52	94.47	2
Fog Node Candidate 7	50.24	40.46	9
Fog Node Candidate 8	59.56	41.12	8
Fog Node Candidate 9	100.00	100.00	1
Fog Node Candidate 10	70.35	36.50	10

The *Fog Node Candidate 9* is ranked in the first place while the *Fog Node Candidate 6* follows with a close score, thus the component recommends Fog Node Candidate 9.

3.4 FUTURE STEPS

The next steps of NebulOuS CC Brokerage include the extension of the proposed mathematical models to accommodate directly criteria measured with qualitative data. Also, we will explore the incorporation of user preferences via the classification of the alternatives, i.e., by the classification of the fog nodes into groups based on indications or perceptions about their performance. Such a prioritization will yield additional weight restrictions that will impose an analogous effect on the Fog Node scores. The new developments will be incorporated in the component as well as open issues regarding the user management, reports with results, etc. will be addressed.

4 BROKERAGE QUALITY ASSURANCE

4.1 INTRODUCTION

Any brokerage service operates based on constraints expressed by the entities interested in using the service for accessing artefacts or assets that are suitable for their purposes. In NebulOuS, these constraints take the form of deployment and runtime requirements posed by stakeholders who are interested in discovering infrastructural fog resources suitable for running application or application component instances.

NebulOuS assures the quality of its fog brokerage service by assessing the quality of the deployment and runtime constraints, henceforth collectively referred to as *application* constraints, upon which this service is based. Deployment constraints impose requirements on the infrastructure over which an application[4] instance is deployed: its compute capacity (expressed as CPU and RAM capacities), its ability to persist data in storage volumes, and its ability to send and receive data over a network. They may also constrain the infrastructure's whereabouts (geolocation), and its provider (i.e., whether an application instance can or cannot be deployed on infrastructure owned or provided by certain entities). Runtime constraints on the other hand impose requirements on the execution of an application instance. They may constrain execution *indirectly*, based on its effect on the underlying infrastructure e.g., CPU utilisation must not exceed 80% of total CPU capacity, but also *directly* e.g., by constraining that the number of frames analysed per second by a surveillance application component must not fall below a preset threshold.

NebulOuS assesses the quality of application constraints by ensuring their compliance with *higher-level meta-constraints*. The latter impose broader-scope QoS requirements that convey an organisation's expectations regarding the consumption of an application. These expectations may be performance ore security driven. As an example, consider an organisation that uses an IoT application. The organisation imposes a QoS meta-constraint whereby application instances must be deployed on infrastructure that features a CPU clock rate of at least 100MHz (e.g., to avoid aliasing effects); moreover, it imposes a security-flavoured meta-constraint whereby the application must never be deployed across infrastructure that is under the control of certain blacklisted (untrusted) organisations. Clearly, any application constraints set by users of this application must abide by these broader organisational expectations.

NebulOuS offers the Brokerage Quality Assurance (BQA) mechanism for ensuring compliance of application constraints with the (organisational) expectations expressed through meta-constraints. The BQA is underpinned by a *semantic model* for expressing both application constraints and meta-constraints. This essentially transforms the process of ensuring constraint compliance into one of *semantic reasoning*, bringing about the following seminal advantages.

(i) Effective reasoning based on knowledge that is potentially semantically inferred and not readily available at the syntactic level. Consider, for instance, a meta-constraint whereby no application instances may be deployed outside the EU. Suppose that an application deployment constraint requires that an application instance is deployed in Athens, Greece. Semantic reasoning allows us to infer that Athens, Greece is indeed in the EU and therefore this application constraint abides with the meta-constraint.

(ii) Reliance on a standards-based approach that avoids potentially error-prone ad-hoc solutions for checking constraint abidance.

4.2 INTERNAL ARCHITECTURE AND INTERFACES

The BQA mechanism comprises two components (see Figure 15): the Meta-constraint Primer and the Application Constraint Feeder. The Meta-constraint Primer interacts with the UI to elicit the knowledge artefacts required for formulating *meta-constraints*; it then uses them to express meta-constraints ontologically and pass them to the Ontology Module (OM). Meta-constraints are *unary* constraints formulated on two arguments: a metric of interest (e.g., CPU cores, CPU utilisation, RAM size, frames per second, IO operations per second, location, etc.), and a corresponding value or value range (the constraint threshold); they also comprise a comparison operator (=, ≠, ≤, <, ≥, >) for comparing metrics against thresholds.

[4] We employ the term "application" to refer both to an application and to an application 'component'. Afterall, an application 'component' is itself an application.

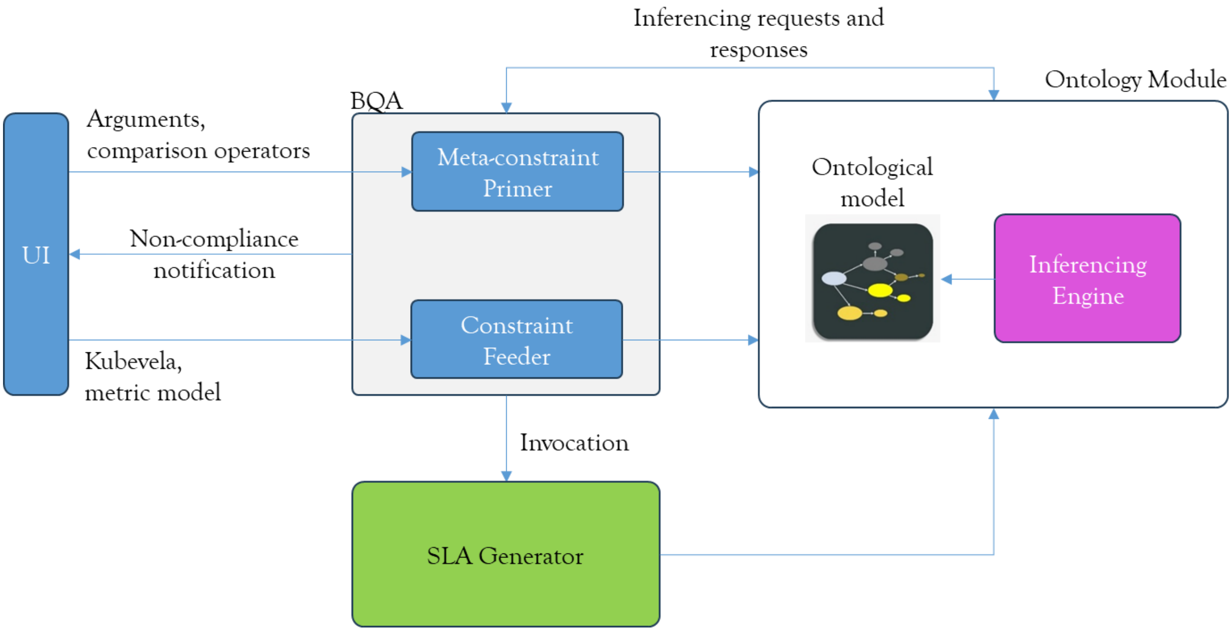


Figure 15: BQA mechanism – internal structure and external interactions

Table 6 illustrates an example of the serialisation format used for representing meta-constraints in interactions between the BQA mechanism and the UI; it also depicts the corresponding ontological representation of meta-constraints (in JSON) passed to the OM.

Table 6. Meta-constraint Primer - format of interaction (example)

UI interaction (YAML)	OM interaction (JSON)

`constraint:`
`  metric: CPU_CORES`
`  operator: '>='`
`  threshold: 4`
	`{`
	`"firstArgument": "CPU_CORES",`
	`"operator": "GREATER_EQUAL_THAN",`
	`"secondArgument": 4`
	`}`

In a similar vein, the Application Constraint Feeder interacts with the UI to elicit the knowledge artefacts required for formulating deployment and runtime constraints; it then uses them to express these constraints ontologically and pass them to the OM. Akin to meta-constraints, deployment and runtime constraints are unary constraints on two arguments and a comparison operator. Deployment constraints are extracted from KubeVela serialisations provided by the UI, and runtime constraints are extracted from the metric model also provided by the UI[5]. Table 7 depicts the serialisation format of deployment constraints and the corresponding ontological representation (in JSON) passed to the OM. The serialisation format of runtime constraints in the metric model is identical to the format used for representing meta-constraints in UI to BQA mechanism interactions (see Table 6) and thus omitted.

[5] More details on KubeVela serialisations and the metric model can be found in [16].

Table 7. Application Constraint Feeder - format of interaction (example)

UI interaction (YAML)	OM interaction (JSON)

`-name: kafka-ui`
`...`
` properties:`
`  ...`
`  cpu: "0.3"`
`  memory: "512Mi"`
`  ...`
	`{`
	`"constraints": [{`
	`    "firstArgument": "CPU_CAPACITY",`
	`    "operator": "EQUALS",`
	`    "secondArgument": 0.3`
	`  },{`
	`    "firstArgument": "RAM_CAPACITY",`
	`    "operator": "EQUALS",`
	`    "secondArgument": 512,`
	`  }]`
	`}`

The BQA mechanism invokes the Inferencing Engine of the OM (see Figure 15) to determine whether the ontological representation of an application constraint (either deployment or runtime) is *compliant* with – i.e., it is *semantically subsumed* by – the ontological representation of a corresponding[6] meta-constraint. If subsumption is indeed inferred, then the application constraint is considered ***compliant*** and it is passed over to the SLA Generator; otherwise, it is considered *non-compliant* and an appropriate notification is emitted. If no corresponding meta-constraint can be found, the application constraint is vacuously considered compliant. Conversely, if a meta-constraint does not correspond to any application constraints, *it is itself converted to an application constraint and passed over to the SLA Generator*. Table 8 depicts an inferencing request sent by the BQA to the OM's Inferencing Engine and the corresponding response received. The request queries the ontology to extract an application constraint and a meta-constraint and determine whether the latter semantically subsumes the former.

Table 8. BQA mechanism - format of interaction (example)

OM interaction (request)	OM interaction (response)

{
"queries": [
"inverse containsConstraint value SPECIFICATION_META_X AND Constraint",
"inverse containsConstraint value SPECIFICATION_APP_X AND Constraint"
],
"action": "validate"
}
	{
	"valid": true
	}
	or
	{
	"valid: false
	justification: "... "
	}

The BQA mechanism invokes the SLA Generator to transform any compliant application constraints, as well as any meta-constraints that do not correspond to any application constraints, into Service Level Objectives (SLOs). Table 9 depicts an example invocation.

[6] A definition of correspondence between application constraints and meta-constraints is provided in Section x.3

Funded by the European Union

Table 9. BQA mechanism - format of interaction (example)

```
{
  "insert": ["SPECIFICATION_APP_X", "SPECIFICATION_META_X "]
}
```

4.3 IMPLEMENTATION

This section outlines our ontological model for capturing application constraints and meta-constraints; it also outlines the process through which the former are assessed for compliance with the latter.

Both application constraints and meta-constraints (henceforth simply referred to as *simple constraints*[7]) are modelled as instances of the OWL-Q class owlq:SimpleConstraint. A simple constraint is associated with its arguments through the properties owlq:firstArgument and owlq:secondArgument, and with its comparison operator through the property owlq:Operator. More specifically, owlq:firstArgument is an object property that associates a simple constraint with a metric i.e., with an instance of the class owlq:Metric. owlq:secondArgument is a data property that associates a simple constraint with the threshold value, or value range, against which the first argument (i.e., the metric) is compared. As an example, Figure 16 illustrates how the application constraint $o \equiv cpuCores = 4$ and the meta-constraint $m \equiv cpuCores \geq 4$ are modelled in OWL-Q.

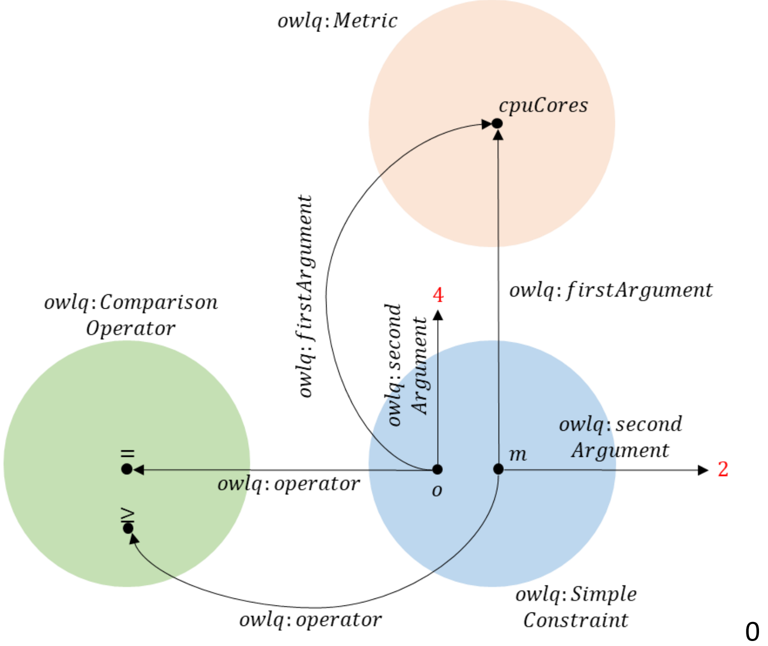


Figure 16: Ontologically capturing application constraints and meta-constraints

[7] Since they are both mathematically unary constraints.

4.4 DETERMINING COMPLIANCE

Let *o* and *m* be an application constraint and a meta-constraint respectively. To determine whether *o* is compliant with *m*, a two-step process is followed. In the first step, *o*'s and *m*'s first arguments are obtained. If they are represented by the same individual from the class owlq:Metric (see Figure 16), then the comparison proceeds to the next step. Otherwise, *o* and *m* are considered *orthogonal* to each other, and *o* is vacuously considered *compliant* with respect to *m*; no further comparison between *o* and *m* takes place.

In the second step, a range of allowable values is calculated for each constraint. This calculation is based on the comparison operator that each constraint features, as well as on its threshold value. For instance, if *o* is the constraint *cpuCores = 4* and m is the constraint *cpuCores ≥ 2*, then the allowable value range for *o* is *[4, 4]* and for *m* is *[2, ∞)*. If *o*'s allowable value range is a subset of *m*'s allowable value range, then *o* is considered *compliant*; in any other case i.e., if there is at least one value in *o*'s range that is not in *m*'s range, then *o* is considered *non-compliant*. All compliant application constraints are passed over to the SLA Generator where they get transformed into SLOs.

In case a meta-constraint is found to be orthogonal to all application constraints for a particular application or application component, then the meta-constraint becomes an application constraint and passed to the SLA Generator.

4.5 STATE-OF-THE-ART AND BEYOND

Several ontology-based formalisms have been proposed for describing QoS constraints. These include: WSAF-QoS [17], DAML-QoS [18], QoSOnt [19], WSMO-QoS [20], OWL-Q [36, 37], onQoS-QL [23], and PCM [24]. Nevertheless, only OWL-Q can be claimed to provide a rich metric model according to the richness criteria in [25][8].

The BQA mechanism advances the state of the art in several ways. Firstly, to the best of our knowledge, there is no prior attempt to assess the quality of a fog brokerage service by ensuring its alignment with higher-level meta-constraints that convey an organisation's broader QoS expectations; an alignment that is achieved by harnessing the application constraints used for delimiting the artefacts or assets that become accessible through the brokerage service. Secondly, the adoption of semantic technologies enables application constraints to be assessed for compliance at the *semantic*, rather than the syntactic, level i.e., based on knowledge that is not syntactically articulated, but semantically inferred, during the reasoning process; this leads to a more efficient quality assurance process. Thirdly, the adoption of semantic technologies enables the articulation of application constraints and meta-constraints that are founded upon custom metrics that can *accurately* convey an organisation's broader QoS expectations. It thus paves the way for a generic and 'malleable' brokerage service that can be shaped according to the particular needs of an organisation.

4.5.1 BQA Mechanism Extensions

In the final iteration (M24-M29 of the project), the BQA mechanism will be further extended with concepts and properties of the Metadata Schema (MDS). The MDS was introduced as part of the Melodic project[9] for addressing multi-clouds requirements and offerings and was extensively updated during the Morphemic

[8] A deeper analysis of these criteria, and a detailed account of OWL-Q, are provided in [16].

[9] https://www.melodic.cloud/

project[10] for coping with additional kinds of resources such as hardware-accelerated resources. Its adoption in NebulOuS will provide a rich vocabulary of terms that will assist the articulation of a wider range of application constraints and meta-constraints. A more detailed account of the MDS can be found in [16].

5 OPTIMISER

5.1 INTRODUCTION

The role of the optimiser is to assign resources to the application components to maximize the *utility* of an application. The various components of the application have various resource requirements like the number of threads a component can use, or the required location, or execution node provider. Furthermore, the distributed application may also have requirements on the multiplicity of components, or vicinity of the component to other application components. Making decisions about these requirements for each component jointly constitutes the *configuration* of the distributed application.

The best application configuration depends on the current circumstances. For example, an application may require more resources during daytime when there are many users of the application or much traffic on the monitored roads, and if the utility of the application includes deployment cost, then the number of application component instances should be decreased and perhaps moved to private infrastructure during the night. Hence, the values measurements taken from the running application will drive the need for reconfiguration.

On the other hand, reconfiguring the application has an overhead as some application microservices may be unavailable during reconfiguration, and adding more resources may take some time before the additional resources are available to the application. Hence, a reconfiguration will only be triggered if the forecasted measurements indicate that one or more of the Service Level Objectives (SLOs) will be violated. The Optimiser will then find the best configuration for the application's forecasted execution context.

Once the optimised configuration has been found for the forecasted execution context, the Optimiser will interact with the Deployment Manager (SAL) to install the application's Kubernetes cluster for the initial deployment, and thereafter modify this cluster throughout the application's lifetime. When the Kubernetes cluster is running, the application's KubeVela file is modified to reflect the parameters of the desired component configuration, and KubeVela will ensure the deployment or reconfiguration or relocation of the application's pods to reflect the optimised configuration.

5.2 INTERNAL ARCHITECTURE AND INTERFACES

The Optimiser Interface deals with the communication to the other components of NebulOuS. The application's KubeVela file and operational constraints are received from the GUI together with the value ranges for the requirement attributes for the application's components, and the utility function to be maximized by the Optimiser. This information is consolidated using the formal optimisation model description using A Mathematical Programming Language (AMPL) [26]. This description is unique for the application to be managed, and the Optimiser Controller will therefore start the application specific components: The Metric Updater, the Performance Module, the Utility Evaluator, and the Solver. The

[10] https://www.morphemic.cloud/

Resource Auction component will be integrated for the second release of the NebulOuS platform. Figure 17 below shows the overall Optimiser architecture.

The Metric Updater receives the list of independent metrics of the application's execution context from the SLO Violation Detector and subscribes to predicted values for these metrics from the Prediction Orchestrator, which is a part of the Event Management System. When the SLO Violation Detector estimates that one or more SLOs will be violated, it sends a reconfiguration event to the Metric Updater, which in turn sends the current execution context to the Solver.

The Utility Evaluator gets the ranked list of available virtualised resources from the Broker, see Section 3 above, and filters the deployment node candidates based on the domains of the requirement attributes of the application components in the optimisation problem. For instance, if no component will be able to use more than 16 GB of memory, all larger virtual machines can be discarded. It prepares a data file to match the optimisation problem for the Solver and updates the data file whenever there is a change in the offered execution possibilities.

The Performance Module maintains regression models allowing the Solver to assess the impact of a changed configuration on the performance indicators used in the utility function. A simple linear regression model will be used in the first release. A digital twin of the running application will be used for the next release to simulate various deployment alternatives and train more advanced and better regression models for the performance indicators.

The Solver components are shown in Figure 18 below. The notable feature is the introduction of the Solver Manager that has been introduced to facilitate the use of the Solver in the training of the SLO Violation Detector. The SLO Violation Detector will need knowledge of the optimal application configurations under various conditions, and it will therefore send application execution contexts to the Solver Manager in the same way as the Metric Updater. The Solver Manager maintains a pool of Solvers and dispatches the received execution contexts to the first available Solver. The AMPL Solver implements a Solver for the AMPL optimisation problem description. The benefit of the AMPL language is that there are a wide range of open source or commercial mathematical solvers accepting AMPL descriptions, and currently it uses the Couenne solver [27]. All Solver components are implemented as independent Actors allowing deadlock free parallel execution [28].

When the Solver has found a set of assignments to the resource attributes for all application components, the corresponding solution will be published by the Solver Manager. This means that the SLO Violation Detector can subscribe to the found solutions and use them for training. Solutions that are resulting from SLO violation events whose application execution context is prepared and sent by the Metric Updater will be marked with a Boolean flag indicating that the found solution should be deployed as a reconfiguration of the running application. These solutions are captured by the captured by the Optimiser Controller and forwarded to the Adapter. The role of the Adapter is to compute the difference between the running configuration and the desired application configuration. The changes to the optimised service graph will then be communicated to the Deployment Manager that will reserve the new virtualised hardware resources with the Broker and add the new services to the application's Kubernetes cluster. When the Adapter gets notified that the new resources are available, it will send a new KubeVela file to the Kubernetes master to reconfigure the application's pods.

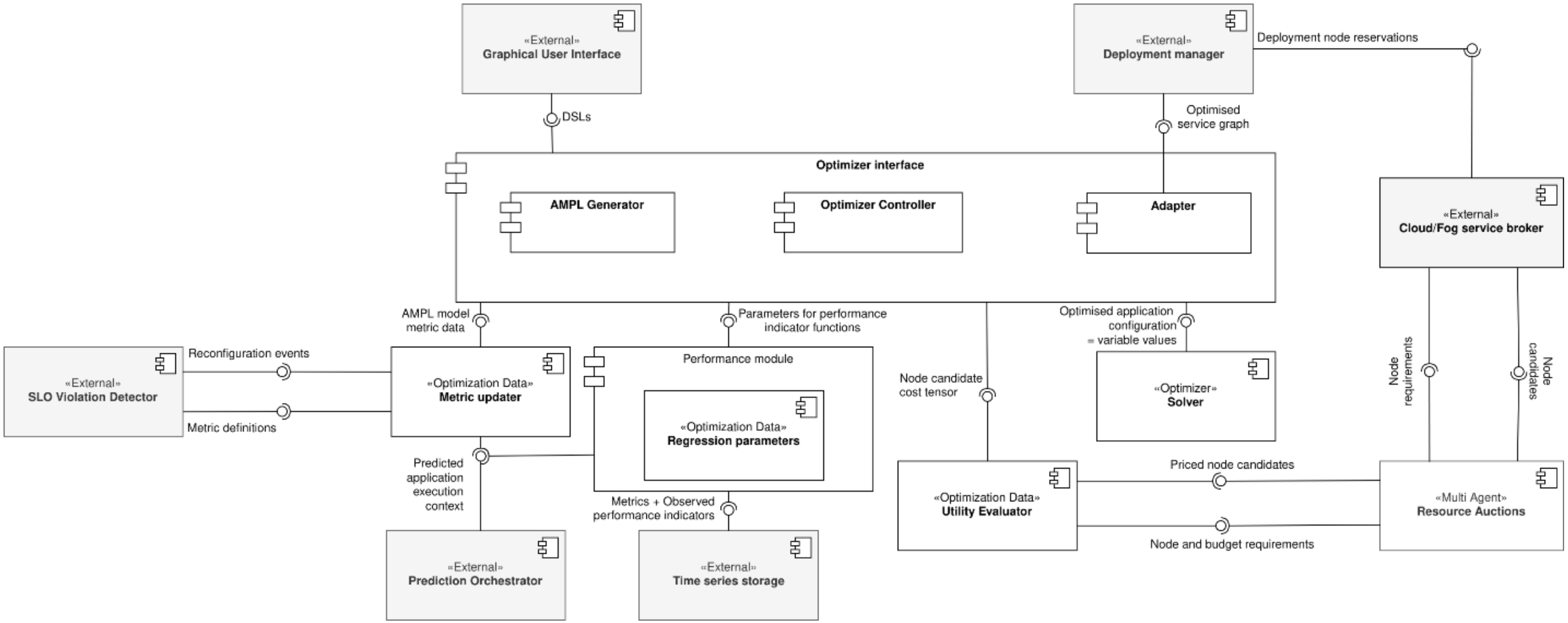


Figure 17: The architecture of the Optimiser module whose components are shown in white interacting with other NebulOuS components indicated in grey.

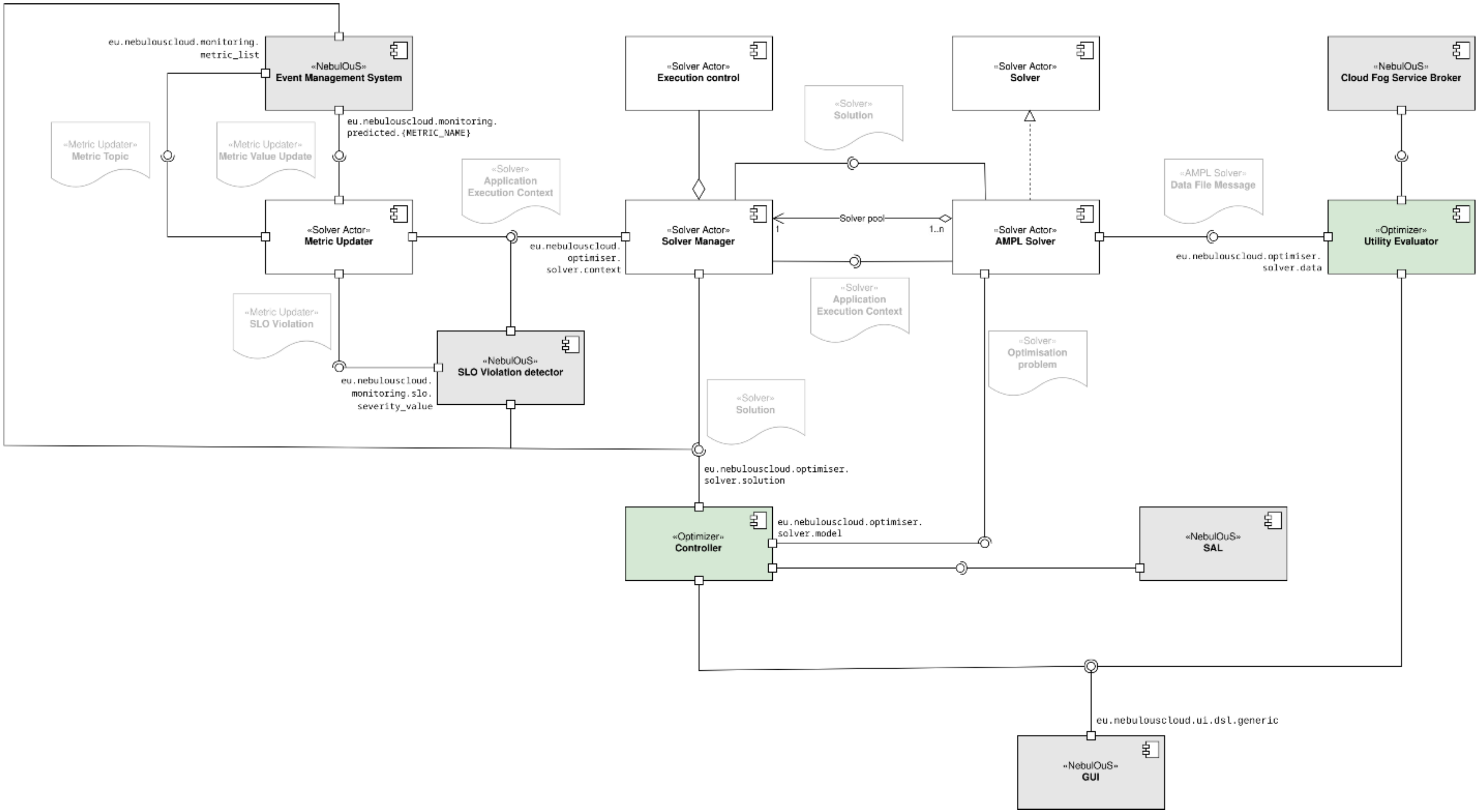


Figure 18: The detailed architecture of the Solver module with the interaction and Advanced Message Queuing Protocol (AMQP) topics used to exchange messages with the other components of the NebulOuS system.

5.3 IMPLEMENTATION

The optimiser-controller component is implemented in Java version 17 and compiles to a "fat jar", i.e., a Java Archive file that includes all dependencies and can be run standalone with only the Java runtime present. The controller component can be run from the command line for testing purposes but is deployed as a container in production. The controller uses the exn-connector-java[11] middleware to communicate via the Advanced Message Queuing Protocol (AMQP[12]) message broker with other NebulOuS components, including the User Interface, the Solver and the Deployment Manager. The main behaviour is implemented by the "NebulousApp" class which is instantiated once per running application. This class is created when the UI starts a new application and manages that application's lifecycle: initial deployment, redeployment, and eventual shutdown. The controller uses the Scheduling Abstraction Layer (SAL) common library[13], created in the MORPHEMIC[10] project, to generate messages towards the execution engine, and the Jackson library[14] to parse JavaScript Object Notation (JSON[15]) and YAML[16] data that arrives from other components. The source code for the optimiser-controller is available in the relevant repository[17] under the NebulOuS code base.

The Utility Evaluator and the Performance Module components are implemented in Java version 17 and for the first release of the NebulOus platform they are packaged into one .jar file and as a single Open Container Initiative (OCI[18]) container. They are implemented as a SpringBoot[19] application and key Spring Framework components[20] are: "PerformanceEstimator", "NodeCandidatesFetchingService", "ExnConnector", "ProactiveConnector", and "NodeCandidateConverter". The dependency management is done with Maven[21]. Both Utility Evaluator and Performance module components use the exn-connector-java[22] middleware to communicate with other NebulOus components via the AMQP. Messages received from other components are parsed with the use of Jackson[23], JSON[24], and sal-common libraries[25]. The source code is available in the NebulOus repository[26].

[11] https://opendev.org/nebulous/exn-connector-java

[12] https://www.amqp.org/

[13] https://github.com/ow2-proactive/scheduling-abstraction-layer/tree/master/sal-common

[14] https://github.com/FasterXML/jackson

[15] https://www.json.org

[16] https://yaml.org/

[17] https://opendev.org/nebulous/optimiser-controller

[18] https://opencontainers.org/

[19] https://spring.io/projects/spring-framework

[20] https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/stereotype/Component.html

[21] https://maven.apache.org/

[22] https://opendev.org/nebulous/exn-connector-java

[23] https://github.com/FasterXML/jackson

[24] https://www.json.org

[25] https://github.com/ow2-proactive/scheduling-abstraction-layer/tree/master/sal-common

[26] https://opendev.org/nebulous/optimiser-utility-evaluator

The Solver components are implemented in the ISO standardised programming language C++[27] using the features of the latest approved standard, C++23. There are 5 classes implemented as Theron++[28] Actors: The Metric Updater; the Execution Control merged with the Solver manager; the abstract Solver implemented by the AMPL Solver, see Figure 18. In Theron++ each Actor is a separate operating system thread, and hence the three Actors are running concurrently exchanging messages. Messages that are received from external components or meant to be consumed by external components are sent as JSON messages using the JSON for Modern C++[29] library on AMQP topics implemented as part of the generic communication layer of Theron++ using the Apache Qpid Proton[30] Application Programming Interface (API) library. Internal messages among Theron++ actors are sent as binary classes, and Figure 18 shows the name of the message classes and which Actor class defining each message type. The code of all Solver components is available in the relevant repository[31] under the NebulOuS code base, and the executable is available as an OCI container.

5.4 STATE-OF-THE-ART AND BEYOND

The main novelty for the first release is the architecture, which is a significant evolution of the architecture used for optimising Cloud applications in the MELODIC[9] and MORPHEMIC[10] projects [29]. The NebulOuS architecture directly implements the proactive Monitor-Analyse-Plan-Execute with Knowledge (MAPE-K) of autonomic computing [30], extended for proactive application management by the researchers in NebulOuS [31].

Another novelty is the way the optimisation problem is formulated and solved. The way an optimisation problem is formulated has much to say for the efficiency of finding a good and optimised solution. The preceding projects use the Cloud Application Modelling and Execution Language (CAMEL) [32]. This Domain Specific Language (DSL) allowed the modelling of the application topology model, the metric mode, and the optimisation problem as one model. Even though this facilitated the coherency of the three models, it also made the language complicated. The consistency of the models in NebulOuS is ensured by the GUI, see Section 2 above. This separation of concerns allows the Optimiser architecture to be significantly simplified, and the optimisation model to be formulated using the standard AMPL language [26], which is directly supported by multiple commercial and open source solvers. The "No Free Lunch" theorem states that there is no universally best solver for all optimisation problems [33], and therefore being able to easily change the underlying solver depending on the problem structure at hand represent a major improvement.

Finally, the architecture has been designed to also support our own solvers, and there are currently investigations ongoing to speed up the search of the variable domains by grouping feasible variable tuples into deployment points for each component. The future inclusion of Digital Twin application models in the Performance Monitor and more advanced machine learning regression models should allow even better and more robust decisions to be made by the solvers.

[27] https://isocpp.org/

[28] https://github.com/GeirHo/TheronPlusPlus

[29] https://github.com/nlohmann/json

[30] https://qpid.apache.org/proton/

[31] https://opendev.org/nebulous/optimiser-solver

6 WORKFLOW EXECUTION

6.1 INTRODUCTION

A single application in Service Oriented Architecture (SOA) should be composed of many independently deployable smaller components. Each component may have different services that communicate with others in various ways, e.g., REST API, Kafka broker or AMQP broker. To reduce development time and associated costs, SOA defines a method for making components more reusable such that each service contains all the code and data integrations needed to perform a complete, discrete function. In addition, the service interfaces provide loose coupling, which means that they are designed as self-contained components that can be called with little or no knowledge of how the integration is implemented beneath.

Moreover, these applications are also expected to run for an extended period under changing application execution contexts and workload; hence they are called *persistent* applications since the application components (functions) are deployed and scaled but remain available as long as there are data items available to process. These discrete requests of data processing are called workload. The requests can be submitted at any time and will be processed by the application's components as they are available until the application's response to the request is finally produced. Therefore, the deployed components may experience idle periods during which they do nothing but wait for the processing to resume depending on the incoming data flow in the application components.

Modelling application as workflows allows the implementation of the SOA concept by dividing a single application (also called a job) into multiple smaller tasks. Each task may be completely independent from the rest and may require or provide a certain service(s) from or to another task(s). In other words, given a discrete request of data processing, a workflow is a set of sequential operations that are started when data is submitted to the initial task, known as the "ingress task," and end when data is removed from the "egress task." The workflow tasks have data dependencies, which result in temporal execution dependencies: Before all of its predecessor tasks have completed processing and the output data from these predecessor tasks is ready for processing, a downstream task cannot begin. As a result, a workflow can be represented as a directed graph that shows the dependencies between the tasks. Since it's frequently assumed that there are no loops, a Directed Acyclic Graph (DAG) is a simpler representation of the workflow. Finally, the workflow scheduler controls the execution of the discrete requests, and it follows up on the progress and the metrics of the actions. The main goal of workflow is to maximize performance by binding computations and data location.

6.2 APPROACH

Workflow scheduling oversees managing and allocating how a certain process is executed within a limited pool of computing resources. This allocation ought to be in line with overarching goals, like minimizing the process's execution time or achieving the best tradeoff between latency and accuracy. In this section, we first explain the idea of a workflow model and how the baseline scheduler is executed. Then, we present how the intelligent scheduler tested on a simulator could improve the baseline scheduler based on the monitoring of computing resources, and lastly, we show how the workflow scheduler is initially integrated with Nebulous and next steps.

The first step consists of defining the application model using KubeVela language where the user can define the application architecture, the deployment requirements and the resource requirements together with the workflow executor component and a communication broker. The workflow executor is responsible for managing and control the workload of discrete workflow requests, e.g., data and a directed graph composed of tasks and dependencies. The tasks are consequently stateless and match application components, which makes them well suited to be implemented by serverless functions or lightweight containers. **¡Error! No se encuentra el origen de la referencia.** shows an example of DAG workflow.

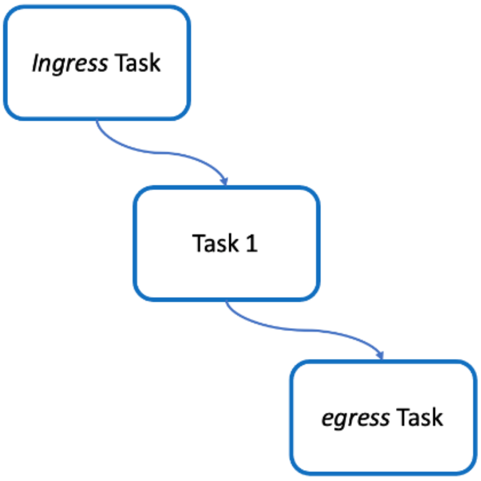


Figure 19: Simple workflow example

In this example, Task 1, which is a computing task, depends on ingress task, which is a preprocessing task. Thus, Task 1 will be executed after ingress Task and egress task will be able to access Task 1's results as output data. For instance, the workflow executor submits a job, the data and the ingress task, to the workflow scheduler, and receives the output data via a communication broker; then the workflow executor submits a new job, task 1 and output of the ingress task, to the workflow. This process is repeated until the egress task is done. Therefore, allocating workflow tasks to the available resources for completion is the responsibility of the workflow task scheduler, a part of the workflow executor. The tasks on the workflow's minimal makespan path and their data dependencies must be taken into account during this allocation process. In order to guarantee that data will be available for the downstream tasks after the predecessor tasks on the minimal makespan path have completed processing, it is also necessary to consider the task duration. The workflow scheduler must either queue up tasks that are ready to be completed or obtain additional resources when the execution resources run out.

The scheduling issue is homogeneous, and the workflow executor has traditionally been a high-performance computing (HPC) data center with identical servers. The unknown task execution times are the source of difficulty. Each task's execution time is therefore a random variate from an observable execution time distribution, and these execution times may vary depending on the data being processed.

Workflows scheduled on a workflow executor within the Cloud continuum will by definition have heterogeneous execution resources, with the resource pool potentially dynamically allocated based on demand. Because the workflow task scheduler is one of the components of the workflow executor application, the scheduling problem is therefore not only a stochastic task allocation problem minimizing the expected makespan but also a simultaneous right-sizing problem for the application.

Given the distributed setup of heterogeneous resources and services distributed across network and computational elements and, it is not trivial to use the existing datacenter resource scheduling technique. For example, Figure 20 shows a deep learning task within a video surveillance application. The variants of the request can be the devices requesting, the accuracy and the latency requirements. Based on these goals and constraints in terms of load spread across the devices, the decision needs to be made at runtime.

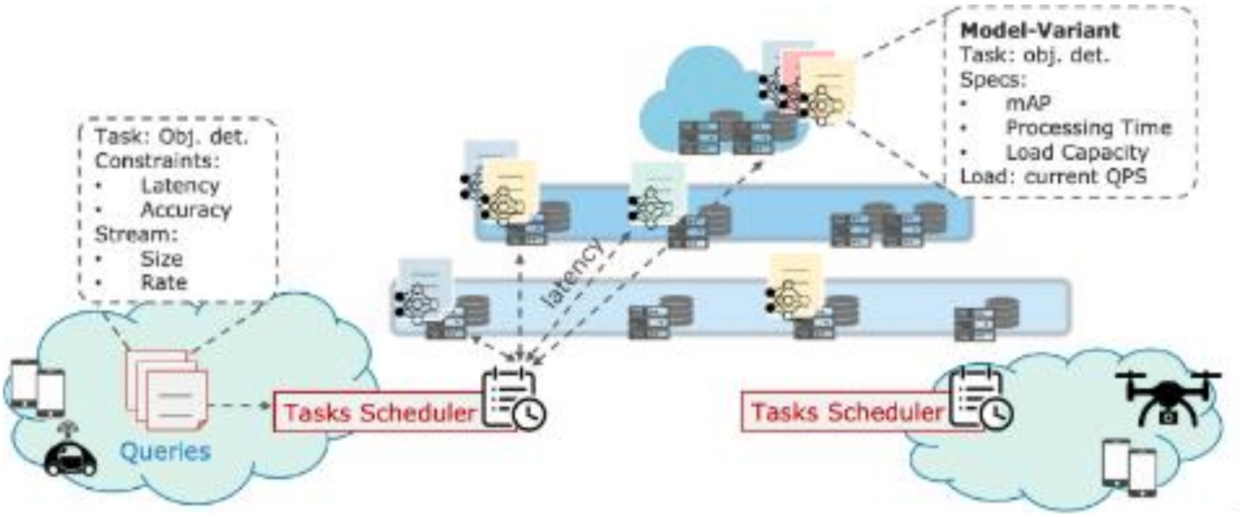


Figure 20: Request scheduling for optimal resource allocation at real time

To this end, we are building a workflow executor as an application where each job, input data and task, is submitted to the workflow scheduler that determines the scheduling policy to distribute the jobs across the available task components. This approach of determining the best scheduling policy will allow to build a learning paradigm based on uncertain network dynamics and algorithms that can learn and adapt their environment based on resource availability. We refer to this as Adaptive Scheduling of Edge Tasks (ASET), wherein a sophisticated reinforcement learning agent, trained on real-world network topology, is used to determine the optimal policy for scheduling workloads by means of Deep Reinforcement learning (DRL) techniques. The policy can be as straightforward as scheduling a task at the closest edge cluster in real time based on load and latency.

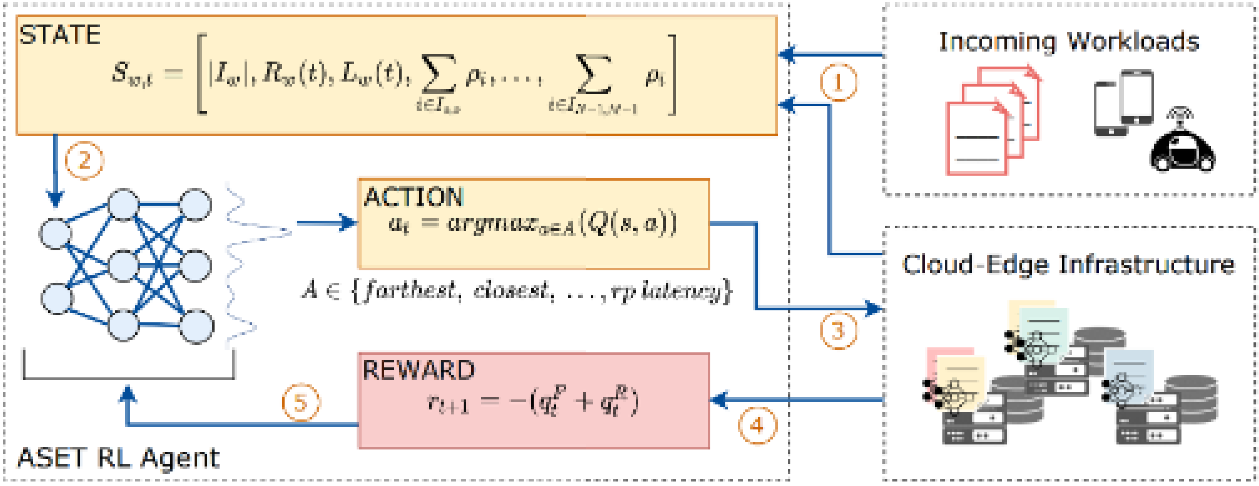

STATE
$S_{w,t} = \left[|I_w|, R_w(t), L_w(t), \sum_{i \in I_{0,0}} \rho_i, \ldots, \sum_{i \in I_{N-1,M-1}} \rho_i \right]$
ACTION
$a_t = argmax_{a \in A}(Q(s,a))$
$A \in \{farthest,\ closest,\ \ldots, rp\ latency\}$
REWARD
$r_{t+1} = -(q_t^P + q_t^R)$
ASET RL Agent
Incoming Workloads
Cloud-Edge Infrastructure
1
2
3
4
5

Figure 21: Adaptive Scheduling of Edge Tasks (ASET) workflow

Our adaptive scheduling method seeks to discover the best course of action based on the state of the system at that moment, including the applications running, the network architecture, and the varying stream arrivals. An intelligent agent attempts to learn the best policy selection strategy based on the observed state of the environment because the optimal policy learning is formulated as a Reinforcement learning (RL) problem due to the lack of labelled data. This is achieved, as illustrated in Figure 21, by an RL policy that estimates a probability distribution of every action that could be taken (policy selection) and cumulatively maximizes a reward (usually maximizing the fraction of successfully served queries). In addition, this approach allows the evaluation of classical scheduling policies.

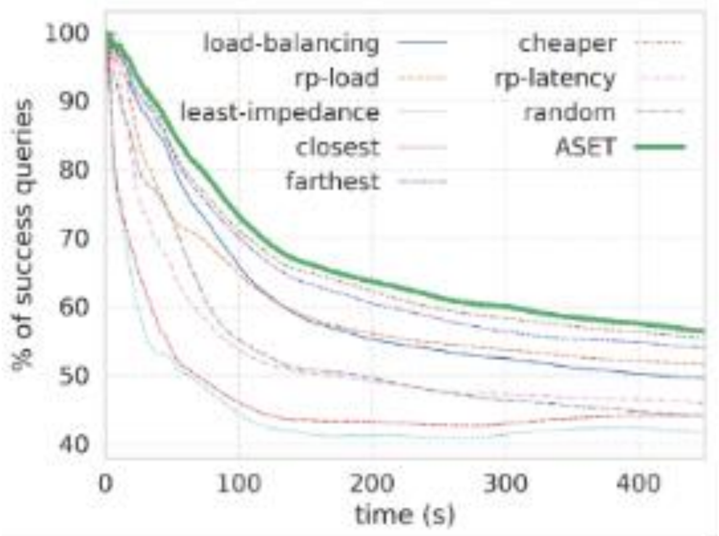


Figure 22: Percentage of successful queries over time for ML task with users arriving in real-world pattern

Preliminary findings on a simulationn environment indicate that ASET outperforms conventional scheduling mechanisms even when only a portion of the network resources are visible. We model the situation where users arrive according to actual patterns. Better policy selection through multi-agent communication, security & privacy-aware, and real-world deployments is still being worked on.

7 CONCLUSIONS

This deliverable has documented the software components of the NebulOuS platform dealing with the *planning* part of the application deployment and reconfiguration. It covers collecting the input from the NebulOuS users in terms of the application topology model, the metric model and information about resource providers and available hardware to be used when configuring and deploying the application. This includes the design and filtering of providers' stated SLOs and the recommendation the alternative. This leads to an optimisation problem that is solved based on information about the application's current execution context to find a configuration optimised for the situation at hand.

The components are now under testing with the NebulOuS pilots and will be further evaluated by the companies successfully responding to the NebulOuS Open Calls. The software will be improved based on the feedback gathered from the evaluations. Furthermore, new modules will be integrated in the next release, most notably the distributed auctions for the resources available to an application, and the live feedback cycle where information about observed SLO violation events will drive the recommendations provided by the Broker.

8 REFERENCES

[1] A. Charnes, W. W. Cooper, and E. Rhodes, 'Measuring the efficiency of decision making units', *European Journal of Operational Research*, vol. 2, no. 6, pp. 429–444, Nov. 1978, doi: 10.1016/0377-2217(78)90138-8.

[2] Zilla Sinuany-Stern, 'Foundations of operations research: From linear programming to data envelopment analysis', *European Journal of Operational Research*, vol. 306, no. 3, pp. 1069–1080, May 2023, doi: 10.1016/j.ejor.2022.10.046.

[3] William W. Cooper, Lawrence M. Seiford, and Kaoru Tone, *Data Envelopment Analysis: A Comprehensive Text with Models, Applications, References and DEA-Solver Software*, Second. New York, NY: Springer US, 2007. doi: 10.1007/978-0-387-45283-8.

[4] Jane Siegel and Jeff Perdue, 'Cloud Services Measures for Global Use: The Service Measurement Index (SMI)', in *2012 Annual SRII Global Conference*, Jul. 2012, pp. 411–415. doi: 10.1109/SRII.2012.51.

[5] I. Patiniotakis, Y. Verginadis, and G. Mentzas, 'PuLSaR: preference-based cloud service selection for cloud service brokers', *Journal of Internet Services and Applications*, vol. 6, no. 1, pp. 1–14, 2015.

[6] Holger Scheel, 'Undesirable outputs in efficiency valuations', *European Journal of Operational Research*, vol. 132, no. 2, pp. 400–410, Jul. 2001, doi: 10.1016/S0377-2217(00)00160-0.

[7] L. Liang, J. Wu, W. D. Cook, and J. Zhu, 'The DEA Game Cross-Efficiency Model and Its Nash Equilibrium', *Operations Research*, vol. 56, no. 5, pp. 1278–1288, 2008.

[8] A. Charnes, W. W. Cooper, and E. Rhodes, 'Measuring the efficiency of decision making units', *European Journal of Operational Research*, vol. 2, no. 6, pp. 429–444, Nov. 1978, doi: 10.1016/0377-2217(78)90138-8.

[9] R. D. Banker, A. Charnes, and W. W. Cooper, 'Some Models for Estimating Technical and Scale Inefficiencies in Data Envelopment Analysis', *Management Science*, vol. 30, no. 9, pp. 1078–1092, 1984, doi: 10.1287/mnsc.30.9.1078.

[10] Tarja Joro, Pekka Korhonen, and Jyrki Wallenius, 'Structural Comparison of Data Envelopment Analysis and Multiple Objective Linear Programming', *Management Science*, vol. 44, no. 7, pp. 962–970, 1998, doi: 10.1287/mnsc.44.7.962.

[11] I. Kaliszewski, 'Out of the mist—towards decision-maker-friendly multiple criteria decision making support', *European Journal of Operational Research*, vol. 158, no. 2, pp. 293–307, Oct. 2004, doi: 10.1016/j.ejor.2003.06.005.

[12] Kaisa Miettinen, *Nonlinear Multiobjective Optimization*. in International Series in Operations Research & Management Science. Springer US, 1998. Accessed: Dec. 10, 2018. [Online]. Available: //www.springer.com/gp/book/9780792382782

[13] R. Allen, A. Athanassopoulos, R.G. Dyson, and E. Thanassoulis, 'Weights restrictions and value judgements in Data Envelopment Analysis: Evolution, development and future directions', *Annals of Operations Research*, vol. 73, no. 0, pp. 13–34, Oct. 1997, doi: 10.1023/A:1018968909638.

[14] Thomas L. Saaty, *The analytic hierarchy process: planning, priority setting, resource allocation*. New York: McGraw-Hill, 1980.

[15] R. G. Thompson, P. S. Dharmapala, E. J. Gatewood, S. Macy, and R. M. Thrall, 'DEA/Assurance Region SBDC Efficiency and Unique Projections', *Operations Research*, vol. 44, no. 4, pp. 533–542, Aug. 1996, doi: 10.1287/opre.44.4.533.

[16] S. Veloudis *et al.*, 'NebulOuS D2.2: Initial Semantic Models and Resource Discovery Mechanism', NebulOuS, Deliverable D2.2, Oct. 2023.

[17] E. M. Maximilien and M. P. Singh, 'A framework and ontology for dynamic Web services selection', *IEEE Internet Comput.*, vol. 8, no. 5, pp. 84–93, Sep. 2004, doi: 10.1109/MIC.2004.27.

[18] Chen Zhou, Liang-Tien Chia, and Bu-Sung Lee, 'DAML-QoS ontology for Web services', in *Proceedings. IEEE International Conference on Web Services, 2004.*, San Diego, CA, USA: IEEE, 2004, pp. 472–479. doi: 10.1109/ICWS.2004.1314772.

[19] G. Dobson, R. Lock, I. Sommerville, and Ian Sommerville, 'QoSOnt: a QoS Ontology for Service-Centric Systems', in *31st EUROMICRO Conference on Software Engineering and Advanced Applications*, Porto, Portugal: IEEE, 2005, pp. 80–87. doi: 10.1109/EUROMICRO.2005.49.

[20] X. Wang, T. Vitvar, M. Kerrigan, and I. Toma, 'A QoS-Aware Selection Model for Semantic Web Services', in *Service-Oriented Computing – ICSOC 2007*, vol. 4749, B. J. Krämer, K.-J. Lin, and P. Narasimhan, Eds., in Lecture Notes in Computer Science, vol. 4749. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 390–401. doi: 10.1007/11948148_32.

[21] K. Kritikos and D. Plexousakis, 'OWL-Q for Semantic QoS-based Web Service Description and Discovery', *Foundation of Research and Technology, Heraklion, Greece*, [Online]. Available: https://publications.ics.forth.gr/_publications/10.1.1.93.9067.pdf

[22] K. Kritikos, D. Plexousakis, and P. Plebani, 'Semantic SLAs for Services with Q-SLA', *Procedia Computer Science*, vol. 97, pp. 24–33, 2016, doi: 10.1016/j.procs.2016.08.277.

[23] G. Damiano, E. Giallonardo, and E. Zimeo, 'onQoS-QL: A Query Language for QoS-Based Service Selection and Ranking', in *Service-Oriented Computing – ICSOC 2007*, vol. 4749, B. J. Krämer, K.-J. Lin, and P. Narasimhan, Eds., in Lecture Notes in Computer Science, vol. 4749. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 115–127. doi: 10.1007/978-3-540-93851-4_12.

[24] F. D. Paoli, M. Palmonari, M. Comerio, and A. Maurino, 'A Meta-model for Non-functional Property Descriptions of Web Services', in *2008 IEEE International Conference on Web Services*, Beijing: IEEE, Sep. 2008, pp. 393–400. doi: 10.1109/ICWS.2008.97.

[25] K. Kritikos *et al.*, 'A survey on service quality description', *ACM Comput. Surv.*, vol. 46, no. 1, pp. 1–58, Oct. 2013, doi: 10.1145/2522968.2522969.

[26] Robert Fourer, David M. Gay, and Brian W. Kernighan, *AMPL - A Modeling Language for Mathematical Programming*, Second edition. Duxbury Press, 2003. [Online]. Available: https://ampl.com/wp-content/uploads/BOOK.pdf

[27] Pietro Belotti, Christian Kirches, Sven Leyffer, Jeff Linderoth, James Luedtke, and Ashutosh Mahajan, 'Mixed-integer nonlinear optimization', *Acta Numerica*, vol. 22, pp. 1–131, May 2013, doi: 10.1017/S0962492913000032.

[28] Gul Abdulnabi Agha, *Actors: a model of concurrent computation in distributed systems.* in The MIT Press series in artificial intelligence. Cambridge, Mass: MIT Press, 1986.

[29] Marta Różańska, Paweł Skrzypek, Katarzyna Materka, and Geir Horn, 'An Architecture for Autonomous Proactive and Polymorphic Optimization of Cloud Applications', in *Proceedings of the 36th International Conference on Advanced Information Networking and Applications (AINA-2022), Volume 3*, Leonard Barolli, Farookh Hussain, and Tomoya Enokido, Eds., in Lecture Notes in Networks and Systems, vol. 451. Conference Location: Sydney, Australia: Springer International Publishing, Apr. 2022, pp. 567–577. doi: 10.1007/978-3-030-99619-2_53.

[30] IBM, 'An architectural blueprint for autonomic computing', IBM, 17 Skyline Drive, Hawthorne, NY 10532, U.S.A., White Paper Third Edition, Jun. 2005. [Online]. Available: http://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf

[31] Marta Różańska and Geir Horn, 'Proactive Autonomic Cloud Application Management', in *Proceedings of the 15th IEEE/ACM International Conference on Utility and Cloud Computing (UCC2022)*, Conference Location: Vancouver, Washington, USA: IEEE/ACM, Dec. 2022, pp. 102–111. doi: 10.1109/UCC56403.2022.00021.

[32] Alessandro Rossini *et al.*, ‘The cloud application modelling and execution language (CAMEL)’, *OPen Access Repositorium der Universität Ulm*, p. 39, Mar. 2017, doi: 10.18725/OPARU-4339.

[33] David H. Wolpert and William G. Macready, ‘No free lunch theorems for optimization’, *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997, doi: 10.1109/4235.585893.

9 APPENDIX: NEBULOUS CC ATTRIBUTE MODEL

All the attributes are analysed in the table below, providing indications on the updates/adjustments provided over SMI and Broker@Cloud models.

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
Accountability: Measures the properties related to the cloud continuum resource provider				Adjusted to Cloud Continuum Resources
	Auditability: The ability of a resource user to verify that the provider is adhering to the standards, processes, and policies that they follow.		Linguistic: { LOW, MEDIUM, HIGH }	Adjusted to Cloud Continuum Resources
	Compliance: It examines whether or not, standards, processes, and policies committed to by the provider, are followed.		Unordered Set	-
	Governance: The processes used by the provider to manage user expectations, issues and perceived performance.		Unordered Set	Adjusted to Cloud Continuum Resources
	Ownership: The level of rights a user has over his/her data, and intellectual property associated with the use of a cloud continuum resource.			Adjusted to Cloud Continuum Resources
	Provider Certifications: The provider maintains current certifications for standards relevant to their user' requirements.		Unordered Set	-
	Provider Contract/SLA Verification: The provider makes available to users SLAs adequate to manage the resource offered and mitigate risks of device unavailability.			Adjusted to Cloud Continuum Resources
	Provider Personnel Requirements: The extent to which provider			Adjusted to Cloud Continuum Resources

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
	personnel have the skills, experience, education, and certifications required to effectively offer and maintain a cloud continuum resource.			
		Technical competency w.r.t resources hardware	Linguistic: { LOW, MEDIUM, HIGH }	Adjusted to Cloud Continuum Resources (Broker@Cloud)
		Technical competency w.r.t network connectivity	Linguistic: { LOW, MEDIUM, HIGH }	Added 3rd Level Attribute
	Provider Supply Chain: The provider ensures that any SLAs that must be supported by its suppliers are supported.			-
	Malfunctions Mitigation Support: The level of provider support in case of resources malfunctions (e.g., High refers to the same day replacement of the faulty device)		Linguistic: { LOW, MEDIUM, HIGH }	Added 2nd Level Attribute
Agility: Indicates the impact of a resource use upon a user's ability to change direction, strategy, or tactics quickly and with minimal disruption.				Adjusted to Cloud Continuum Resources
	Adaptability: The ability of the provider to adjust to changes in user requirements.		Linguistic: { LOW, MEDIUM, HIGH }	-
	Elasticity: The ability of adjusting the offered resource capacity to meet demand (e.g., in cases that a fog resource had been partially made available to NebulOuS).			Adjusted to Cloud Continuum Resources (Broker@Cloud)
		Time	Range in seconds	
		Extend offered processing capacity	Boolean	Added 3rd Level Attribute
		Extend offered memory capacity	Boolean	Added 3rd Level Attribute

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
		Extend offered network capacity	Boolean	Added 3rd Level Attribute
	Extensibility: The ability to add new features or services to existing offered resources.			Adjusted to Cloud Continuum Resources
		Geographic Coverage (number of available locations in the world)	Integer	Added 3rd Level Attribute (adjusted from (Niemcewicz, 2021))
	Flexibility: The ability to add or remove predefined features from a resource.			Adjusted to Cloud Continuum Resources
	Portability: The ability of a user to easily move a service from one provider to another with minimal disruption.		Linguistic: { LOW, MEDIUM, HIGH }	-
	Scalability: The ability of a provider to increase or decrease the number of cloud continuum resources offered in a certain area to meet client requirements.			Adjusted to Cloud Continuum Resources
		Cloud resources addition	Boolean	Added 3rd Level Attribute
		Fog resources addition	Boolean	Added 3rd Level Attribute
		Edge resources addition	Boolean	Added 3rd Level Attribute
		Total number of available Fog resources	Integer	Added 3rd Level Attribute
		Total number of available Edge devices	Integer	Added 3rd Level Attribute
	Moveability: The likelihood that a certain fog or edge resource leased to be used by an external user, can be moved away from its original physical location, during the contracting period. This can have either a positive (resource move based on the application			Added 2nd Level Attribute

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
	requirements) or negative (unforeseen resource move) impact on the application and its agility.			
		Desired Move Support	Boolean	Added 3rd Level Attribute
		Unforeseen Move Likelihood	Linguistic: { LOW, MEDIUM, HIGH }	Added 3rd Level Attribute
Assurance: Indicates how likely it is that the service will be available as specified				Adjusted to Cloud Continuum Resources
	Availability: The amount of time that a user can make use of a service.		Percentage	-
	Maintainability: It refers to the ability for the provider to make modifications to the resource to keep the offereing in a condition of good repair.		Linguistic: { LOW, MEDIUM, HIGH }	-
	Recoverability: It is the degree to which a resource is able to quickly resume a normal state of operation after an unplanned disruption.			Adjusted to Cloud Continuum Resources (Broker@Cloud)
		Recovery Time	Range in seconds	-
	Reliability: It reflects measures of how a resource operates without failure under given conditions during a given time period.			-
		Uptime	Seconds or Percentage	Adjusted to Cloud Continuum Resources
	Resiliency/Fault Tolerance: The ability of a resource to continue to operate properly in the event of a failure.			Adjusted to Cloud Continuum Resources
	Service Stability: The degree to which the resource is resistant to (network) change, deterioration, or displacement.		Linguistic: { LOW, MEDIUM, HIGH }	Adjusted to Cloud Continuum Resources (Broker@Cloud)

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
	Serviceability: The ease and efficiency of performing maintenance and correcting problems with the resource.			Adjusted to Cloud Continuum Resources (Broker@Cloud)
		Free support	Boolean	-
		Type of support (hours/days/response time)	Unordered Set { e.g. Basic - 24x7x24, Bronze - 24x7x4, Silver - 24x7x2, Gold - 24x7x1, Platinum - 24x7x½, Gold - 24x7x1 (Reseller)}	-
		Support satisfaction	Linguistic: { VERY LOW, LOW, MEDIUM, HIGH, VERY HIGH, PERFECT }	-
		Network Support	Unordered Set	Added 3rd Level Attribute
Financial: It is related to all the economic aspects related to using a cloud continuum resource				Adjusted to Cloud Continuum Resources
	Billing Process: The level of integration that is available between the user and provider's billing systems and the predictability of periodic bills.			-
	Cost: The user's cost to exploit a cloud continuum resource over time. This includes the cost of data migration, along with recurring costs (e.g., monthly access fees) and usage based costs.			Adjusted to Cloud Continuum Resources (Broker@Cloud)
		Operation cost	Float (€/hour)	-
		Data-Inbound cost	Integer (GB/month)	-

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
		Data-Outbound cost	Integer (GB/month)	-
		Storage	Integer (GB)	-
	Financial Agility: The flexibility and elasticity of the financial aspects of the provider's resources			-
	Financial Structure: How responsive to the user's needs are the provider's pricing and billing components		Linguistic: { LOW, MEDIUM, HIGH }	-
Performance: It covers the features and functions of the provided resources				Adjusted to Cloud Continuum Resources
	Capacity: The maximum number of resources that a provider can deliver while meeting agreed SLAs.			Adjusted to Cloud Continuum Resources (Broker@Cloud)
		Number of CPU Cores	Integer	
		CPU MFLOPs	Float	Added 3rd Level Attribute
		Clock Speed	Float (GHz)	-
		Number of GPU Cores	Integer	-
		GPU MFLOPS	Float	-
		Memory Size	Integer (GB)	-
		Memory Speed	Integer (MHz)	-
		Storage Capacity	Integer (GB)	-
		Storage Throughput	Integer (MB/s or IOPS)	-
		Solid State Drive	Boolean	Added 3rd Level Attribute

Funded by the European Union

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
	Accuracy: The extent to which a resource adheres to its requirements.		Linguistic: { LOW, MEDIUM, HIGH }	-
	Network Functionality: The specific features provided by a resource.			Adjusted to Cloud Continuum Resources (Broker@Cloud)
		Features	Unordered Set (e.g, Bluetooth/LoRa WAN/Zigbee connectivity)	-
		Bandwidth	Float (Mbps)	Added 3rd Level Attribute
		Upload Speed	Float (Mbps)	Added 3rd Level Attribute
		Download Speed	Float (Mbps)	Added 3rd Level Attribute
		Network Throughput	Ineteger (Mbit/s)	-
	Suitability: How closely do the capabilities of the resource match the needs of the user.			Adjusted to Cloud Continuum Resources
		Proximity to Data Source	Float (Km)	Added 3rd Level Attribute
		Proximity to POI (point/area of interest defined by the user)	Float (Km)	Added 3rd Level Attribute
Security and Privacy: It indicates the effectiveness of a provider's controls on access to resources, data, and the physical facilities from which resources are provided				Adjusted to Cloud Continuum Resources
	Access Control & Privilege Management: Policies and processes in use by the provider to ensure that only the provider and user personnel with appropriate role/reasons to access a resource may do so.			Adjusted to Cloud Continuum Resources (Broker@Cloud)

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
		Supported authentication schemes	Unordered Set {Basic, 1-way SSL, 2-way SSL}	-
		Role based Access Control(RBAC) supported	Boolean	-
		Attribute based Access Control supported(ABAC)	Boolean	Added 3rd Level Attribute
	Data Geographic/Political: The user's constraints on resource location based on geography or politics.			-
	Data Integrity: Keeping the data that is created, used, and stored in its correct form so that users may be confident that it is accurate and valid.			-
	Data Privacy & Data Loss: User restrictions on access and use of data are enforced by the provider. Any failures of these protection measures are promptly detected and reported to the user.			Adjusted to Cloud Continuum Resources (Broker@Cloud)
		Audit Trailing	Boolean	-
	Physical & Environmental Security: Policies and processes in use by the provider to protect the provider facilities and physical resources from unauthorized access, damage or interference.			Adjusted to Cloud Continuum Resources
	Proactive Threat & Vulnerability Management: Mechanisms in place to ensure that the resource is protected against known recurring threats as well as new evolving vulnerabilities.			-
		Firewall (UTM-unified threat management)	Boolean	-
	Retention/Disposition: The provider’s data retention and disposition processes meet the users' requirements.		Linguistic: { LOW, MEDIUM, HIGH }	-

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
	Security Management: The capabilities of providers to ensure network, data and infrastructure security based on the security requirements of the user.			Adjusted to Cloud Continuum Resources
		Encrypted Storage	Boolean	-
		Encryption Type	Integer (e.g. 128, 192, 256 Bits)	-
		Transport Layer Security	Boolean	Added 3rd Level Attribute
	Process Transparency: This attribute refers to the availability of open source code, open source business processes and open source hardware from the cloud continuum provider side		Boolean	Added 2nd Level Attribute
Usability: It is related to the ease with which a resource can be used				-
	Client Personnel Requirements: The minimum number of personnel satisfying roles, skills, experience, education, and certification required of the user to effectively access and utilize a resource.			-
	Installability: Installability characterizes the time and effort required to get a resource ready for use.		Linguistic: { LOW, MEDIUM, HIGH }	-
	Learnability: The effort required of users to learn to access and use the resource.		Linguistic: { LOW, MEDIUM, HIGH }	-
	Operability: The ability of a resource to be easily operated by users.		Linguistic: { LOW, MEDIUM, HIGH }	-
	Transparency: The extent to which users are able to determine when changes in a feature of the resource occur and whether these changes impact usability.		Linguistic: { LOW, MEDIUM, HIGH }	-

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
	Understandability: The ease with which users can understand the capabilities and operation of the resource.		Linguistic: { LOW, MEDIUM, HIGH }	-
	Reusability: How generic the resource interface is and how easy it is to be used in different cloud applications		Linguistic: { LOW, MEDIUM, HIGH }	Adjusted to Cloud Continuum Resources (Broker@Cloud)
Reputation: It is related to the reputation of the provider and of the cloud continuum resource offerings that are provided or that have been provided in the past.				Adjusted to Cloud Continuum Resources (Broker@Cloud)
	Brand name: It is the linguistic expression of the reputation of the provider as perceived by its users		Linguistic: { BAD, OK, GOOD }	-
	Resource Reputation: It is the linguistic expression of the reputation of the cloud continuum resource as perceived by its users		Linguistic: { BAD, OK, GOOD }	Adjusted to Cloud Continuum Resources (Broker@Cloud
	Provider Trust It refers to the linguistic expression of the level of confidence that the provider is and will continue to abide to legal security and be compliant with local regulations		Linguistic: { LOW, MEDIUM, HIGH }	Added 2nd Level Attribute (adjusted from (Niemcewicz, 2021))
	Contracting Experience: Indicators of users effort and satisfaction with the process of entering into the agreements required to use a resource.		Linguistic: { LOW, MEDIUM, HIGH }	Moved from other 1st Level Attribute (i.e. Accountability)
	Ease of doing business: Users' satisfaction with the ability to do business with a certain provider.		Linguistic: { LOW, MEDIUM, HIGH }	Moved from other 1st Level Attribute (i.e. Accountability)
	Provider business stability: The likelihood that a certain		percentage (fuzzy number)	Moved from other 1st Level Attribute (i.e. Accountability)

1st Level Attributes	2nd Level Attributes	Indicative 3rd Level Attributes	Indicative Value Types	Update on SMI/Broker@Cloud
	provider will continue to exist throughout the contracted term.			
	Resources stability: The likelihood that certain resources, contracted to be exploited by an external user will not be moved and will continue to exist throughout the contracted term.		percentage (fuzzy number)	Added 2nd Level Attributes
	Provider Ethicality: Ethicality refers to the manner in which the provider conducts business; it includes business practices and ethics outside the scope of regulatory compliance. It also includes fair practices with suppliers, customers, and employees		Linguistic: { LOW, MEDIUM, HIGH }	Moved from other 1st Level Attribute (i.e. Accountability)
	Sustainability: The impact on the economy, society and the environment of a certain provider.			Moved from other 1st Level Attribute (i.e. Accountability) & Added 3rd Level Attributes
		Economic impact	Linguistic: { LOW, MEDIUM, HIGH }	-
		Societal impact	Linguistic: { LOW, MEDIUM, HIGH }	-
		Energy consumption	Float (Watts)	-
		Carbon footprint	Float (g/KWh)	-
	Provider Track record: The previous experience and performance history with respect to leasing cloud continuum resources from a certain provider			Added 2nd Level Attribute
	Resource Track record: The previous experience and performance history with respect to using certain cloud continuum resource types from a certain provider			Added 2nd Level Attribute

CONSORTIUM

NebulOuS

A META OPERATING SYSTEM FOR BROKERING
HYPER-DISTRIBUTED APPLICATIONS ON
CLOUD COMPUTING CONTINUUMS

Funded by
the European Union

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or Directorate-General for Communications Networks, Content and Technology. Neither the European Union nor the granting authority can be held responsible for them.