Università degli Studi di Milano

Dipartimento di Informatica
"Giovanni degli Antoni"

Dottorato di Ricerca in Informatica
XXXVIII Ciclo

Settore Scientifico Disciplinare INF/01

Smart Data-Intensive Service Composition in Cloud-Edge Continuum

Tesi di Dottorato di Ricerca di:
Ruslan Bondaruc
Matr. R13905
ORCID n. 0000-0002-2581-9915

Relatore:
Prof. Marco Anisetti

Correlatore:
Prof. Claudio Agostino Ardagna

Coordinatore del Dottorato:
Prof. Roberto Sassi

Tesi redatta con il contributo finanziario dell'Unione europea - Next Generation EU e Leonardo S.p.A.
CUP n. G43C22002330004

A.A. 2024/2025

Abstract

The proliferation of IoT devices, real-time analytics, and data-intensive applications has challenged the adequacy of traditional cloud computing, which, despite its scalability, often fails to meet the stringent latency, bandwidth, and reliability requirements of modern services. Edge computing partially addresses these needs by bringing computation closer to data sources, but its inherent heterogeneity and limited resources restrict its applicability. The emerging Cloud-Edge Continuum promises to unify cloud and edge infrastructures, enabling dynamic placement and migration of services across heterogeneous environments. However, this paradigm also introduces new research challenges concerning efficient service placement, security-aware deployment, expressive workflow modeling, and scalable optimization.

Existing approaches to service orchestration in the continuum remain limited in several respects. Service placement is widely acknowledged as an NP-hard problem, and most heuristic or mathematical solutions do not scale to large and dynamic deployments. Security and vulnerability considerations are often absent from placement models, despite their direct impact on system stability and migration costs. Traditional workflow description languages were designed for centralized systems and lack the ability to capture runtime adaptability, polymorphic deployment, and secure annotations required in heterogeneous environments. Reinforcement learning has recently emerged as a promising direction for placement optimization, yet many studies rely solely on simulations, fail to integrate multi-objective and security constraints, and lack real-world validation. These gaps motivated the research conducted in this thesis, which aims to enable smart, secure, and adaptive service composition in the Cloud-Edge Continuum.

The thesis presents four main contributions. First, it introduces a QoS-aware deployment framework that models services and facilities using annotated graphs and automatically generates deployment recipes while ensuring that non-functional requirements such as latency, confidentiality, and integrity are preserved. Second, it proposes a vulnerability-aware placement and migration methodology that incorporates vulnerability scoring and migration-aware cost functions into placement decisions, mitigating the risks of instability and recursive migrations. Third, it develops a polymor-

phic workflow description language (cWDL), which extends existing workflow models with QoS annotations and choreography constructs, enabling adaptive, secure, and polymorphic deployment across heterogeneous infrastructures. This language is supported by a deployment engine that translates abstract workflows into executable artifacts for diverse platforms such as Kubernetes and Docker. Finally, the thesis explores reinforcement learning as a placement optimization strategy, formulating the problem as a Markov Decision Process and applying advanced deep reinforcement learning techniques, notably Maskable PPO, to achieve multi-objective optimization under security constraints.

Extensive experimental evaluations validate these contributions. The QoS-aware framework demonstrates efficient deployment across heterogeneous infrastructures. The vulnerability-aware methodology achieves superior security and stability compared to existing approaches. The cWDL improves both representational efficiency and runtime adaptability over traditional workflow models. The reinforcement learning approach significantly reduces missed deployments, overprovisioning, and resource imbalance, achieving near-optimal placement in large-scale scenarios.

Overall, the thesis advances the state of the art in Cloud-Edge Continuum computing by providing a unified set of frameworks, methodologies, and languages for QoS-aware, secure, and intelligent service deployment. By addressing critical gaps in placement scalability, vulnerability integration, and workflow adaptability, it lays the foundation for trustworthy and efficient service orchestration across heterogeneous infrastructures, bringing the continuum paradigm closer to practical adoption in data-intensive domains such as smart healthcare, autonomous systems, and critical infrastructures.

Acknowledgements

I would like to take this opportunity to thank all those who have been part of my PhD journey and who have helped me bring it to completion.

First and foremost, I would like to thank my supervisors. To Marco Anisetti, for giving me this opportunity, for teaching me dedication and instilling in me a passion for research and computer science, for guiding and supporting me, and for his patience throughout this journey. To Claudio Agostino Ardagna, for his constant support and for going out of his way on many occasions to help me. I owe it to both of you that I have had the privilege of writing these pages.

I would also like to thank the people with whom I have had the chance to collaborate. My thanks to Filippo Berto, Rémi Badonnel, Nicolas Schnepf, Alberto Ovena, Mattia Perfumo, and Fabio Ferrari for their contributions to this work.

A special thank you also goes to the students, researchers, staff, and professors who have filled the SESAR Lab over these years. To Antongiacomo Polimeno, Jonatan Maggesi, Nicola Bena, Alex Della Bruna, Marco Luzzara, Samira Maghool, Fulvio Frati, Valerio Bellandi, Mattia Occhipinti, Fatemeh Mohammadi, Mahboubehsadat Jazayeri, and Annalisa Barsotti for sharing lunches, moments, and laughter, and for making this journey lighter. Thanks as well to Antonio Sciarappa, Nicola Ferrarese, Michele Russo, and all my colleagues at Leonardo for the wonderful period in Genoa.

Lastly, a heartfelt thank you goes to my family. Mom, Dad, Giorgio, Giulia, thank you for always standing by me, encouraging me, believing in me, and pushing me to work hard and persevere. Thanks also to Andrea, a lifelong friend, for helping me through difficult times and for sharing a home with me.

Contents

List of Figures

List of Tables

1 Introduction

The rapid proliferation of IoT devices, mobile applications, and real-time analytics has led to unprecedented volumes of data and increasingly stringent requirements on latency, reliability, and security. Traditional cloud computing, with its elasticity and scalability, has been the dominant paradigm for the past decade. However, the centralization of resources in remote data centers makes it ill-suited for latency-sensitive or bandwidth-intensive applications, such as augmented reality, autonomous driving, and smart healthcare, where decisions must be made in milliseconds [30, 138].

Edge computing addresses some of these issues by bringing computation closer to the data source, thereby reducing latency and enhancing context-awareness. Yet, edge infrastructures are inherently heterogeneous and resource-constrained [31, 121]. The emerging Cloud-Edge Continuum paradigm seeks to integrate the strengths of both cloud and edge computing by offering a unified, heterogeneous infrastructure where services can be dynamically placed and migrated according to application requirements [15, 133].

1.1 Motivations

Despite its promise, the Continuum introduces new challenges. First, service placement is an NP-hard problem, requiring trade-offs between latency, bandwidth, cost, and energy consumption [21]. Heuristic and mathematical optimization techniques exist but often fail to scale in dynamic, large-scale deployments [69]. Second, security and vulnerability management re-

main underexplored in placement research. While some works consider secure service orchestration (e.g., SecFog [73], SecFaaS2Fog [32]), few explicitly integrate vulnerabilities into placement decisions, despite their impact on migrations and system stability. Concerning the definition of applications, workflow description languages (WDLs), such as BPEL or BPMN, were designed for centralized systems and lack support for runtime adaptability, secure annotations, and polymorphic deployment across heterogeneous infrastructures [93, 128]. Finally, Reinforcement learning (RL) has emerged as a powerful approach to tackle placement complexity. Deep Reinforcement Learning (DRL) methods now account for nearly 30% of AI-based placement studies [95], but most works are limited to simulation-based evaluations and lack integration with security and multi-objective optimization [10, 44, 48]. These open issues motivate the research in this thesis. To fully exploit the potential of the Cloud-Edge Continuum, there is a need for frameworks, models, and languages that support QoS-aware, secure, and adaptive service deployment, and for intelligent optimization strategies capable of operating effectively in large-scale, dynamic environments.

1.2 Objectives

The overarching objective of this thesis is to enable smart data-intensive service composition and placement in the Cloud-Edge Continuum. The specific goals are:

- **QoS-Aware Deployment.** Design a methodology and framework for deploying services across heterogeneous infrastructures that preserves non-functional requirements such as latency, confidentiality, and integrity.
- **Vulnerability-Aware Placement.** Develop models and strategies for incorporating vulnerabilities and migration costs into placement decisions, mitigating risks of instability and recursive migrations.
- **Polymorphic Workflow Description Language (cWDL).** Propose a workflow language that extends traditional WDLs with annotations,

choreography constructs, and polymorphic deployment capabilities, bridging the gap between abstract design and executable workflows.

- **Reinforcement Learning for Placement.** Explore RL techniques for service placement, modeling it as a Markov Decision Process (MDP) and employing DRL algorithms to optimize deployments under multiple objectives, including security.

1.3 Contributions of the Thesis

This thesis makes four main contributions targeting the gaps identified in the state of the art and addressing the objectives in Section 1.2. The contributions are detailed in the rest of this section.

1.3.1 QoS-Aware Deployment Framework

We present a framework that models workflows and continuum facilities using annotated graphs and performs requirement-resource matching to generate executable deployment recipes. The framework supports heterogeneous infrastructures (cloud, telco-edge, on-premises) and ensures that non-functional requirements are preserved. Experimental evaluation confirms its ability to handle realistic workloads efficiently.

1.3.2 Vulnerability-Aware Placement and Migration

We propose a methodology that integrates vulnerability models into placement decisions. By introducing vulnerability scoring, cost functions, and migration-aware optimization, our approach improves both security and resource efficiency compared to existing methods. Contributions include a formal system model, placement strategies that mitigate recursive migrations, and validation through extensive simulation.

1.3.3 Polymorphic Workflow Description Language (cWDL)

We extend WDLs with QoS annotations and choreography constructs, enabling workflows to adapt to heterogeneous facilities and dynamic contexts.

The cWDL is supported by a Polymorphic Deployment Engine (PDE), which interprets annotated workflows, injects support services when necessary (e.g., for integrity enforcement), and generates executable deployment artifacts, such as Kubernetes, Docker Compose, AWS Step Functions. Experimental evaluation shows improvements in representation efficiency and runtime performance over traditional orchestration models.

1.3.4 Reinforcement Learning for Placement

We model service placement as an MDP and design a reinforcement learning methodology leveraging Maskable PPO, enabling efficient training and multi-objective optimization. Contributions include the formal definition of states, actions, and rewards for placement; the integration of security constraints into the learning process; and a large-scale evaluation that demonstrates significant reductions in missed deployments, overprovisioning, and resource imbalance.

1.4 Organization of the Thesis

The thesis is organized as follows.

Chapter 2 reviews the state of the art in Cloud-Edge Continuum, covering service placement, workflow description languages, and reinforcement learning approaches.

Chapter 3 presents the proposed QoS-aware deployment framework, including methodology, architecture, and experimental evaluation.

Chapter 4 introduces vulnerability-aware deployment strategies, integrating vulnerability models and recursive migration mitigation mechanisms.

Chapter 5 describes the polymorphic workflow description language (cWDL), its methodology, architecture, and validation.

Chapter 6 develops the reinforcement learning methodology for service placement, including the MDP formulation, RL algorithms, and performance evaluation.

Chapter 7 summarizes the main findings, highlight limitations, and outline directions for future research.

2

Foundations and State of the Art

The evolution of distributed computing has been deeply influenced by the growing need to handle massive volumes of data while ensuring low latency, scalability, and resilience. Traditional cloud computing, while offering virtually unlimited resources, often suffers from limitations in latency, bandwidth, and context-awareness when faced with modern data-intensive applications. To address these challenges, the Edge-Cloud Continuum (ECC) has emerged as a paradigm that seamlessly integrates edge, fog, and cloud resources, enabling the flexible deployment of services across heterogeneous infrastructures. This paradigm shift has attracted significant research attention, particularly around the problem of service placement, which is widely recognized as an NP-hard optimization challenge involving constraints on latency, energy, costs, and security.

Alongside placement, research has focused on workflow description languages (WDLs), which enable the specification, coordination, and execution of complex service compositions across distributed environments. Traditional orchestration-centric models, exemplified by BPEL, have gradually given way to choreography and more adaptive languages, designed to better support decentralized, dynamic, and QoS-aware deployments. More recently, reinforcement learning (RL) has emerged as a powerful approach to tackle the inherent complexity of service placement in the ECC, with deep reinforcement learning (DRL) methods showing particular promise in handling high-dimensional, dynamic, and model-free optimization problems.

This chapter provides an overview of the state of the art on service placement and related foundations in the Edge-Cloud Continuum. It first introduces the notion of the Edge-Cloud Continuum and its main architectural and operational characteristics (Section 2.1). It then surveys service placement approaches, including application-driven requirements, algorithmic models, and optimization objectives, while highlighting open research gaps (Section 2.2). Next, it discusses workflow description languages for modeling and coordinating service compositions (Section 2.3). Finally, it explores reinforcement learning solutions applied to service placement, focusing on algorithms, objectives, and challenges (Section 2.4), thereby setting the stage for the framework and methodologies developed in this thesis.

2.1 Edge-Cloud Continuum

The Edge-Cloud Continuum has emerged as a fundamental paradigm shift, integrating decentralized edge resources with centralized cloud infrastructure to meet the increasing demand for low-latency, scalable, and end-to-end service delivery [29]. Driven primarily by the exponential growth of data generated by Internet-of-Things (IoT) devices and the necessity for real-time responsiveness [29, 133], the continuum orchestrates computation across the entire Internet-scale path, moving processing progressively closer to data producers while still leveraging cloud-scale capacity [29]. This model fundamentally challenges the traditional centralized cloud computing paradigm, which often struggles with high data volumes and long distances, leading to significant communication latencies and bottlenecks [170]. Although the ECC is widely adopted, the literature presents a fragmented landscape with varying terminologies, often being referred to interchangeably as the cloud continuum, cloud-edge continuum, IoT-edge-cloud, or cloud-to-things continuum [97, 117]. Fundamentally, the cloud continuum is defined as an extension of the traditional Cloud toward multiple entities (e.g., Edge, Fog, IoT) that provide analysis, processing, storage, and data generation capabilities [117].

The architectural foundation of the ECC is typically described as a hierarchical, multi-layered structure designed to optimize data storage, processing, and analysis while ensuring low-latency communication [30, 88].

This architecture can be viewed from a cloud-centric perspective, classifying layers based on their proximity to the central public cloud data centers (Cloud layer) [34]. Progressing toward the data source, the Near Edge layer consists of regional or in-country mini data centers, which still operate in a multi-tenancy mode and facilitate faster data processing before reaching core cloud infrastructure [39]. The Far Edge layer places computing resources closer to end-user environments (e.g., phone towers, industrial sites), handling localized computing tasks [39]. Moving further down, the On-Premise layer uses private edge nodes (e.g., local facilities) very close to devices, ensuring autonomous and real-time connectivity [1]. The closest layer to the data source is the On-Device layer, encompassing IoT devices, sensors, cameras, and smartphones, which perform preliminary tasks like filtering and aggregation to reduce latency and network overhead [30, 1].

A rigorous analysis of characteristics and performance metrics is essential for designing efficient and scalable ECC systems [42]. Latency is arguably the most critical metric, achieving ultra-low performance ($<$ 1ms) at the on-device and on-premise layers, but steadily rising as processing moves toward the distant cloud layer (potentially exceeding 20ms) [49]. The ability to accommodate new devices, users, and services, or scalability, is conversely correlated with distance: the on-device layer exhibits very high scalability in accommodating endpoints (millions), while the cloud layer generally displays the lowest scalability in adding new physical nodes [43]. Similarly, computation capabilities scale from very low at the device layer (suitable for basic tasks) to very high in centralized cloud infrastructures, enabling large-scale big data processing and AI-driven analysis [53]. Furthermore, ECC layers differ in resource management based on tenancy and privacy: on-device and on-premise layers typically use dedicated infrastructure, yielding high levels of privacy, while far-edge, near-edge, and cloud layers adopt multi-tenancy models, which may present varying, often lower, degrees of privacy protection [55, 56].

The success of the ECC hinges on leveraging appropriate paradigms and models for computation, communication, and deployment [57]. Computational paradigms distribute data analytics, typically relying on the edge for tasks like filtering and aggregation to reduce data volume before transmission to the cloud [59, 56]. Modern demands frequently require Distributed

Learning, where collaborative paradigms like Federated Learning (FL) enable multiple clients to train a global model while keeping their data decentralized, addressing crucial issues such as data privacy and transfer minimization [61, 66]. Similarly, Split Learning (SL) partitions the model itself across clients, ensuring data privacy while optimizing for resource-constrained devices [155]. Regarding communication models, the traditional Client-Server model is extended in multi-tier architectures where edge layers act as local servers for latency-sensitive tasks. Essential for IoT networks is the Publish-Subscribe model, implemented by lightweight protocols such as MQTT [120] and AMQP [171], which facilitate asynchronous, scalable data dissemination. Newer protocols like HTTP/3, built on QUIC, are also becoming important due to their ability to maintain active connections and provide low latency in dynamic cloud-edge environments [141, 101].

Deployment strategies allow for the flexible execution of workloads across heterogeneous infrastructure [71]. Containerization (e.g., Docker) represents a particularly lightweight form of virtualization, offering isolation, faster startup times, and lower resource overhead compared to traditional Virtual Machines (VMs), making it highly suitable for resource-constrained edge environments [75, 68]. The Serverless deployment (Function-as-a-Service, FaaS) paradigm is gaining traction as it abstracts underlying infrastructure management, billing users only for execution time [148, 145]. While FaaS functions can dynamically scale and migrate across layers to adapt to network conditions and resource availability [75, 148], many existing FaaS frameworks rely on centralized or cloud-centric architectures, complicating efficient geo-distributed deployment at the edge [181]. To counter performance degradation caused by resource limits and high latency, performance optimization techniques are critical: task offloading shifts computationally intensive tasks from resource-limited devices to higher-powered servers to reduce latency and save energy [89], often utilizing sophisticated AI-driven algorithms (e.g., reinforcement learning) for real-time, dynamic decision-making [142]. Furthermore, service caching strategies reduce latency and congestion by storing frequently accessed content closer to end-users, typically relying on predictive models or collaboration among caching entities [127]. Major public cloud providers (e.g., AWS, Azure, Google Cloud) actively support this ecosystem by offering edge-specific enabling services

(e.g., AWS Local Zones, Azure Stack Edge) to extend cloud-like capabilities across the continuum [97].

Despite its potential, adopting and implementing the ECC faces numerous challenges and open issues. A major hurdle is the vast heterogeneity and interoperability resulting from the diversity of devices, platforms, architectures, and protocols involved, exacerbated by the lack of universally standardized protocols and mature development frameworks [62]. Resource management and orchestration across geographically distributed nodes presents a complex optimization problem, requiring systems capable of real-time adaptation to dynamics like device mobility, fluctuating network conditions, and energy constraints [160]. Ensuring security and privacy is crucial due to the expanded attack surface at the edge and the complexity of establishing trust among distributed devices owned by different entities, mandating robust encryption and privacy-preserving techniques like federated learning [8, 127]. Other key concerns include optimizing energy efficiency and sustainability for battery-powered edge devices [99], and achieving coherent data management and distributed analytics, which involves maintaining data consistency between fast, local edge processing and long-term cloud storage [79]. Looking forward, the emergence of advanced AI models, notably Generative AI and AI Agents, is expected to shape the future of ECC by enhancing autonomous decision-making and developer tooling, despite the substantial computational and complexity challenges these integrations introduce [55].

2.2 Service Placement

2.2.1 Application Characteristics

The placement optimization problem in edge computing is profoundly influenced by the nature and requirements of the deployed applications, which are categorized based on their intended use case and service structure [37, 140, 183]. Common application types driving edge development include Internet of Things (IoT), Industry 4.0, Big Data Analytics, and Telco NFV-based services [183]. IoT applications typically involve continuous data streams from numerous sensors, requiring placement solutions capable of handling

arbitrary latency constraints, varying bandwidth profiles, energy optimization (to prolong device battery life), and supporting dynamic mapping for mobility (e.g., in intelligent transportation) [129, 70]. Industry 4.0 applications, such as cloud robotics, impose strict Quality of Service (QoS) requirements for real-time operation and mission-critical systems, often necessitating additional orchestration features like anti-affinity rules to deploy replicas to physically separate servers, thereby guaranteeing dependability and reliability [182]. Big Data Analytics can leverage hybrid edge/cloud environments to perform data processing closer to the source, reducing network load and achieving low end-to-end processing latency, which is crucial for time-critical stream analytics applications like computer vision, often requiring responses within 100 ms [185, 164]. Telco services, often modeled as Service Function Chains (SFCs) composed of Virtual Network Functions (VNFs), must meet stringent Service Level Agreements (SLAs) regarding availability, reliability, delay, and mobility for a large customer base [164]. Service structure further defines the placement complexity, distinguishing between single-component monolithic services (often seen in offloading scenarios) and multi-component services [146, 5]. A sophisticated variant of multi-component services, which is addressed by recent research, explicitly models connections/relations between components by imposing network requirements, such as latency budgets or network traffic characteristics [146, 110]. While placement research is critical to performance, only a small fraction of studies formally incorporate security and privacy features, addressing concerns such as preventing location or usage pattern privacy leakage induced by wireless task offloading [118, 69, 86], or integrating specific security parameters as formal constraints [157].

2.2.2 Algorithmic Models and Solutions

The core mathematical challenge of optimal placement involves abstracting applications or services into entities that require assignment to available infrastructure [153]. This selection process, which involves matching entities to places, often couples the placement decision with offloading, resource provisioning, and scheduling [153]. Consequently, these complex, multi-faceted problems necessitate diverse mathematical frameworks and solving

approaches [153, 119].

Linear Programming (LP) and Nonlinear Programming (NLP) are the most widely used formalizing frameworks, both falling under the category of mathematical programming [91]. LP aims to optimize a linear objective function subject to linear constraints, while NLP includes constraints or objective functions that are nonlinear [91]. The nature of placement problems, which deal with the arrangement of discrete structures such as tasks and hosts, makes combinatorial optimization the most prominent formalization choice [91]. Within this domain, several formulations are frequently encountered. Integer Linear Programming (ILP) restricts all decision variables to integer values and is commonly applied to capture discrete placement decisions, such as assigning tasks to hosts or routing flows along network paths [91]. Mixed-Integer Linear Programming (MILP) extends this approach by combining integer and continuous variables [91]. An illustrative MILP formulation of Service Graph Embedding (SGE) defines an objective function that minimizes bandwidth allocation costs, node allocation costs, and the inverse of the minimum resource load for load balancing, while simultaneously enforcing constraints to ensure every service node is mapped to a suitable resource node, resource capacities are respected, flow preservation is guaranteed, and strict latency bounds are upheld for service chains [168, 108, 134, 63, 126]. More expressive formulations, including Integer Nonlinear Programming (INLP), Mixed-Integer Nonlinear Programming (MINLP), and Mixed-Integer Quadratically Constrained Programming (MIQCP), extend the integer-based models to nonlinear or quadratically constrained objective functions and constraints, enabling the representation of complex performance metrics, nonlinear resource interactions, and quadratic flow or interference conditions [91].

For applications involving multiple connected components, especially VNF service chains, models based on Graph Theory are commonly used, transforming the problem into Service Graph Embedding (SGE) or Virtual Network Embedding (VNE) [58]. These frameworks map a service graph, where nodes represent functions and edges denote requirements, onto a resource graph representing the physical infrastructure [58]. VNE problems, however, are highly complex; most practical variants are NP-hard and strongly inapproximable [19, 147].

Beyond these, other theoretical frameworks have also been adopted in research [83]. Constraint Programming offers a generic and adaptive modeling approach well suited to dynamic fog computing environments where objectives and constraints may change [5, 83, 90]. Stochastic Optimization is employed when uncertainty plays a significant role, such as in random task arrivals or user mobility [83]. Within this category, techniques such as Optimal Stopping Theory support sequential delay minimization [83, 7], Multi-armed Bandits address partially known optimization rewards [83], and Stochastic Knapsack Problems model constraints with random item sizes [83]. Optimal Control Theory represents another important framework, focusing on determining optimal control laws over time in dynamic systems [83]. Examples include Lyapunov optimization, which is used to obtain asymptotic optima, stabilize parameters such as battery capacity [187, 60], and support location prediction [110, 109], as well as Job shop scheduling [83]. Finally, Game Theory provides a means of modeling strategic interactions among decision-makers, such as task owners and host providers [83, 47]. Within this framework, Potential Games [111, 47], Matching Games [64, 63], and Stackelberg Games for sequential decision-making [187, 184] are widely explored, often with the aim of achieving Nash equilibrium [111, 187, 184].

Because most placement problems are NP-hard, the majority of solutions rely on custom heuristics rather than purely analytical methods or commercial solvers [116]. Among the most pervasive approaches are heuristic and search techniques [116, 177]. For VNE and SGE, one of the dominant methods is the heuristic-guided greedy backtracking search [125, 159], which iteratively attempts to map service components (“legs”) onto substrate resources [125]. When constraints are violated, the algorithm backtracks by undoing the most recent greedy step and exploring stored alternatives in the search space [125, 159]. The efficiency of this method depends on the branching factor, the number of stored alternatives, and the backtracking depth [159]. Iterative heuristics [69, 123] and Shortest Path Search particularly, when scheduling is represented in a time slot graph [177, 176], are also employed. Complementing these, decomposition techniques partition complex problems into smaller, more manageable sub-problems [116]. For example, Benders decomposition separates offloading optimization into a master problem and sub-problems that can be solved in parallel [173], while other

works decompose the problem across architectural layers such as cloud, fog, and terminals, applying different solving methods to each [54, 168].

Population-based approaches, such as evolutionary and meta-heuristics, are equally popular for tackling complex optimization tasks [177]. Genetic Algorithms (GAs), often combined with Monte Carlo simulations, are widely applied to find near-optimal placements by iteratively improving solution fitness against objectives such as QoS, cost, and utilization [36, 19, 186]. Other meta-heuristics, including the Sine Cosine Algorithm [83] and simulated annealing [189, 106], are also reported in the literature. Finally, several specialized mathematical methods are applied in narrower contexts. Matching theory is often used to address resource assignment sub-problems, with one-to-one or one-to-many models suited to decentralized computation offloading and resource allocation [83, 137], while the Hungarian method is employed for optimal assignments [54, 23]. Column generation is an effective tool for large-scale ILP or MILP formulations, where it produces partial solutions that reduce execution times and yield approximation algorithms with performance guarantees [82, 124, 175]. In distributed environments, trading games and matching problems have been leveraged to design distributed algorithms for resource assignment and orchestration, providing efficiency in multi-actor contexts [137, 92].

The optimization objectives in these models define the desired outcomes, while constraints formalize platform limits and application requirements [163, 188]. Four major objectives dominate the literature, and they often appear in combination [174]. The first is delay or processing time minimization, the most widespread goal, reflecting the prevalence of latency-sensitive applications and the importance of meeting low-latency QoS requirements [20, 162]. A second common objective is energy minimization, particularly relevant in mobile edge scenarios where end devices or edge nodes such as UAVs operate under tight battery constraints. This goal is frequently optimized jointly with delay minimization [111, 174, 162]. A third area of focus involves revenue maximization or cost minimization, which reflects the service provider's perspective by prioritizing either profit maximization through efficient resource utilization or cost reductions through optimized operation [174, 162, 96]. The fourth major objective is resource utilization maximization, which is often pursued alongside revenue maximization. In

some cases, however, resource utilization is deliberately minimized to preserve capacity for future applications and to safeguard long-term revenue streams [174]. Other objectives are less frequently studied but nonetheless important; these include maximizing throughput, reducing bandwidth usage, maximizing coverage, or enhancing QoS aspects such as reliability, often achieved by minimizing the number of redundant backups [41].

The constraints that shape optimization are equally diverse. Computation resources, including CPU, memory, and storage, represent the most commonly applied set of restrictions across the surveyed literature [188]. Network capacity constraints are also prominent, reflecting the scarcity of bandwidth in edge networks [188]. Many models further include strict delay bounds, which are especially critical for latency-sensitive applications even when delay is not the primary optimization objective [163, 188]. In the specific context of MEC, constraints related to radio resources and mobility are widely applied; these cover parameters such as signal-to-noise ratio, wireless channel capacity, and the migration of users or mobile edge nodes like UAVs [163, 100]. Finally, a range of additional constraints appears in some formulations, including dependability requirements such as affinity and anti-affinity rules to enforce or forbid component collocation, location tags, explicit cost limits, and—more rarely—security and privacy parameters like trust levels or user classes [182, 179].

Overall, models that incorporate utilization in their objective functions tend to require more sophisticated solving techniques, suggesting that approximating utilization optimization usually demands advanced and specialized heuristics [92].

2.2.3 Research Gaps

Several critical areas require further investigation to enable fully functional, dynamic edge platforms [132].

A significant research gap concerns enhancing awareness to user mobility in edge placement decisions [132]. Combining classical user mobility models with modern AI-based prediction techniques is a promising area to develop proactive hybrid placement strategies that could greatly improve resource provisioning effectiveness and service QoS in dynamic systems [132].

Closely related is the observation that no related work has, to the best of the authors' knowledge, considered exploiting smart cars, particularly electric cars during charging, as temporary edge nodes to augment infrastructure capacity dynamically, using predicted paths and locations as input for infrastructure planning [46].

Another shortfall is the lack of cross-layer optimization with edge application design [45]. Cloud management frameworks rarely expose their underlying placement policy (such as the optimization target function or overall system status) via an API to application owners [45]. A deeper integration allowing applications to adapt their parameters based on expected placement decisions could lead to more efficient system operation [45].

The trend toward Function-as-a-Service (FaaS) or serverless computing in edge environments is rarely addressed [22]. Given that FaaS functions are often ephemeral and stateless, relying on external data/state, researchers advocate for the emergence of joint placement policies of functions and their corresponding data/state in edge systems to mitigate serious QoS degradation caused by remote data access [22, 184]. This also applies to Coded Distributed Computation (CDC), where the joint placement optimization of computation functions and their input data files is essential to avoid high data reallocation loads [184].

The integration of dynamics and resilience is also a challenge [112]. The highly complex optimization problem of integrating placement methods with real-time auto-scaling capabilities (e.g., Kubernetes) or reliability measures (necessary due to edge infrastructure errors) necessitates research into temporal placement policies to provide essential end-to-end latency guarantees [112, 166].

A pervasive omission is the inadequate attention paid to security-aware edge placement challenges [136]. Security and privacy aspects are often neglected during system design [136, 60]. Future placement engines must incorporate security and privacy requirements, such as trust orientation for IoT services or differential privacy manipulations, into the core orchestration logic, recognizing that edge nodes are often untrustworthy, especially in public multi-provider setups [60, 84].

Gaps also exist in the economic aspects of edge placement [7]. While some works focus on revenue maximization for the provider, there is a com-

plex and largely unsolved research gap in the cost optimization of application owners, which should span the application's whole lifetime: from design choice (monolith vs. microservice), service model (VM, container, FaaS), and provider selection, with placement being the focal point of this challenge [138]. The interaction between user pricing strategy and provider profit maximization is also an active area of research [7].

Finally, the lack of solutions addressing the multi-operator setting is identified as the most common shortcoming [152]. The vast majority of research assumes single-operator scenarios, but global-scale service deployments require compute and network resources from multiple federated providers [152]. Combining placement optimization with economic modeling in this multi-stakeholder environment is a critical research direction [152].

2.3 Workflow Description Languages

Workflow description languages constitute a crucial area of software engineering research, providing formalisms to specify how multiple services or activities are composed to achieve complex goals, traditionally segregated into centralized and decentralized interaction models. The classical dichotomy distinguishes between orchestration, where a single central coordinator dictates the flow and interactions among services, exemplified by languages like Business Process Execution Language (BPEL) [128, 50], and choreography, which adopts a decentralized, peer-to-peer approach detailing collaborative message exchanges from a global viewpoint [21, 128, 76]; this distinction is formally recognized in standards such as the Web Services Choreography Description Language (WS-CDL) and Choreography Diagrams within the Business Process Modeling Notation 2.0 (BPMN2) [128, 21, 50]. Early workflow systems often relied on notations like high-level Petri nets, which, while providing formal semantics, exhibited suitability challenges when handling intricate control-flow requirements necessary for non-trivial workflow processes, specifically struggling with patterns such as those involving multiple instances (especially without a priori knowledge), advanced synchronisation mechanisms (like the synchronizing merge), and non-local cancellations (like global timeouts or cancellation patterns) [169]. To address these shortcomings while retaining the formal foundation of Petri

nets, Yet Another Workflow Language (YAWL) was proposed, extending the notation with explicit constructs, such as the OR-join, the capacity to remove tokens from selected parts of the workflow using the rem function, and dedicated parameters (nofi) to manage the creation, synchronization, and termination of multiple task instances [169]. Beyond structural enhancements, other specialized languages have emerged to address distinct computational contexts; for example, the Context-Aware Workflow Language (CAWL) was designed for ubiquitous computing environments to overcome the limitations of older languages (like BPEL4WS) whose static XML linking functions could not adequately express dynamic context information and to explicitly support the combination of multiple individual workflows into composite services, addressing the necessity for multi-workflow support in pervasive environments where numerous users and services coexist [50]. Similarly, addressing representational efficiency in process modeling, the Partially Ordered Workflow Language (POWL) introduced a hierarchical structure combining partially ordered graphs with process tree operators ($\times$ for exclusive choice and $\square$ for do-redo loops) to model concurrent and sequential behavior while preserving the guaranteed soundness often associated with hierarchical models, achieving greater simplicity compared to complex Workflow nets (WF-nets) [98].

More recently, research efforts have concentrated on choreographic programming, a paradigm aiming for compliance-by-construction, exemplified by platforms such as CHOReVOLUTION, which synthesizes the distributed coordination logic needed to enforce global collaboration specified by a choreography, sparing developers from writing complex interaction code [21]. Building upon previous choreographic systems like Chor and AIOCJ, the object-oriented language Choral introduced a fundamental advancement by generalizing data types to include multiple locations, represented as $T@(A1, \ldots, An)$, enabling the abstract definition of choreographies as classes whose objects and behaviors are collaboratively implemented by specified roles [76]. This object-oriented interpretation allows Choral to integrate mainstream abstractions, support modularity by generating complaint-by-construction libraries for each role, and tackle the critical knowledge of choice problem through mechanisms like the @SelectionMethod annotation, ensuring roles agree on chosen alternative behaviors [76]. Furthermore, the

necessity for workflow languages to support runtime changes is paramount, especially within dynamic infrastructures like cloud computing, where managing service composition is complex due to fluctuating QoS parameters and changing service providers [93, 128]; consequently, languages such as YAWL, AIOCJ, and the coordination language Reo are notable for explicitly supporting runtime adaptability or dynamic reconfiguration to meet these inherent challenges [128].

2.4 Reinforcement Learning for Service Placement

The deployment of services, such as virtual network functions (VNFs) and microservice modules, within dynamic and resource-constrained environments like Mobile Edge Computing (MEC), Fog Computing (FC), and the ECC presents significant management challenges [10, 78, 103, 95]. Service placement is recognized as an NP-hard optimization problem, characterized by fluctuating resource availability, stochastic user demands, and dynamic network connections [10, 104, 107, 151]. Given these complexities, traditional optimization techniques and heuristic methods often prove insufficient or too slow for online operation [10, 78, 103, 107]. Reinforcement Learning (RL), particularly in its deep variant (DRL), has emerged as a crucial methodology for service placement due to its inherent characteristics of self-learning and self-adaptation, which are ideal for modeling high-dimensional, model-free problems [10, 95, 139]. Research indicates that RL accounts for a large portion of machine learning solutions in this field, with DRL methods making up nearly 30% of the relevant studies [95]. The objective across these studies is typically framed as a Markov Decision Process (MDP) or a finite-steps MDP, allowing an intelligent agent to learn an optimal policy π^* that maximizes the long-term cumulative reward by selecting sequences of placement actions [104, 151, 48, 107, 103].

2.4.1 DRL Algorithms and Specialized Architectures

The selection of the appropriate DRL algorithm is dictated largely by the nature of the action space inherent in the placement problem. For environments modeled with discrete action spaces, such as deciding whether to de-

ploy a task or selecting a specific location, value-based methods like Deep Q-Networks (DQN) remain widely utilized for their robust effectiveness and relatively low implementation complexity [107, 151, 139, 130, 78, 95]. Variants like Double DQN and Dueling DQN are also employed to enhance stability and performance, mitigating issues such as Q-value overestimation [78, 139, 130]. Conversely, many sophisticated placement scenarios require determining both placement location (discrete) and the allocation of fractional resources (continuous), necessitating the use of policy gradient methods or hybrid models [104, 44, 95]. Deep Deterministic Policy Gradient (DDPG) and its Twin-Delayed (TD) variant are prominent choices for continuous action spaces, aiming to find policies that maximize expected returns, sometimes incorporating mechanisms like the proposed weight-averaged twin-Q-delayed (WATQD) method to reduce Q-estimation bias and improve training stability [44, 48, 130]. To seamlessly handle the coupling of discrete placement and continuous resource allocation, hybrid solutions such as the Parameterized Deep Q-Network (PDQN) are employed [104]. Policy-based methods like Proximal Policy Optimization (PPO), including its maskable variant (Maskable PPO), and the asynchronous Advantage Actor-Critic (A3C) algorithm are also leveraged due to their enhanced stability, superior performance in complex multi-objective scenarios, and suitability for parallel training architectures [139, 130, 80, 178, 95].

Beyond core algorithms, significant effort is placed on architectural innovations to efficiently process the complex input data structures inherent in networking problems. Specialized neural networks are integrated to enhance the agent's perception: Graph Neural Networks (GNNs), such as Graph Convolutional Networks (GCNs), are crucial for mining the non-Euclidean structural features of physical network topologies and the intricate dependency graphs within modern microservice applications [44, 178, 95]. Additionally, recurrent architectures like Long Short-Term Memory (LSTM) networks are embedded within DRL agents to capture the temporal behavior and dynamic patterns of inputs, which is particularly valuable in environments with stochastic traffic or communication delays [78, 80, 95]. Furthermore, to address variable input and output sizes, such as those arising from Service Function Chains (SFCs) with varying lengths or Joint Caching and Computing Service Placement (JCCSP) problems, encoder-decoder models based on

sequence-to-sequence learning are employed, often using Pointer Networks to realize sequence-to-sequence mapping for service placement actions [48, 31, 178, 95].

2.4.2 Optimization Objectives and Scope of Placement

DRL-based service placement aims to maximize the long-term discounted cumulative reward, which is formulated to achieve optimal performance across several critical metrics, often involving multi-objective trade-offs [104, 48, 130, 103, 31]. The primary objectives commonly addressed include minimizing latency (such as end-to-end delay, processing delay, response time, and task waiting time) [104, 44, 107, 31], reducing energy consumption and overall monetary cost (e.g., deployment cost and resource consumption cost) [44, 80, 103, 95], and maximizing performance indicators like Quality of Service (QoS), Quality of Experience (QoE), and the Service Acceptance Ratio (SAR) [103, 31, 107, 130]. Load balancing is a core optimization target, ensuring balanced workloads and maximizing resource utilization across heterogeneous network resources, particularly in distributed environments [107, 103, 44]. When objectives are contradictory, for instance, minimizing latency often competes with reducing resource congestion, multi-objective DRL methods aggregate objectives, sometimes employing scalarization techniques like the Chebyshev scalarization approach to efficiently explore the non-dominated solution space (Pareto Fronts) [31, 130].

The scope of DRL applications spans multiple granularities of placement problems. Service Function Chain Placement (SFCP) focuses on mapping ordered sequences of VNFs to physical nodes to satisfy resource and delay constraints [103, 31, 178]. DRL-SFCP approaches leverage GCNs and sequence-to-sequence models, often using the A3C algorithm for training, to generate placement strategies that significantly improve acceptance ratio and resource revenue [178]. For microservice (MS) placement, solutions like Dual-GNN Deep Deterministic Policy Gradient (DG-DDPG) integrate GNNs to model the application's internal structure, enabling joint optimization of both placement location and fractional resource allocation for MS modules, even when dealing with continuous action spaces and hard constraints [44, 95]. Furthermore, DRL is applied to the complex JCCSP problem for sensing-

data-driven IoT applications, using encoder-decoder models to orchestrate the joint placement of service functions and multiple computing functions while navigating enormous state and action spaces [48]. Finally, placement concerns extend even to the underlying infrastructure, addressing the Edge Server Placement Problem (ESPP) by modeling placement location sequences as states and movement decisions as actions, utilizing DQN to minimize access delay and balance server workloads [107].

2.4.3 Strategies for Practical Implementation and Scalability

To enhance performance stability, accelerate convergence, and ensure scalability in highly distributed and stochastic environments, researchers propose sophisticated architectural and training strategies. The challenge of long learning times and high initial exploration errors in DRL agents, which result in poor Quality of Service (QoS) during initial deployment, is addressed through proactive mechanisms [151]. One key strategy is the implementation of a two-phase learning process involving bootstrapping: an initial policy is trained offline, either using historical data (such as server logs) or through simulation models, to generate a "mature" model that is then deployed to the online environment (like the orchestrator node) [151, 139]. This approach enables the agent to start making proactive, efficient placement decisions before demands actually occur, thus avoiding detrimental exploration mistakes [151].

To manage large-scale networks and improve learning efficiency, distributed DRL frameworks are widely employed [78, 80]. Distributed DRL agents, such as those in the HOODIE scheme, are deployed at each edge server to autonomously make task placement decisions with only local observability, minimizing the need for complex global coordination and reducing inference delay [78]. Specialized distributed architectures based on IMPALA (IMPortance weighted Actor-Learner Architectures) are also utilized for dynamic application placement of complex Directed Acyclic Graph (DAG) applications, where multiple actors generate diverse experience trajectories in parallel and leverage V-trace off-policy correction to reduce exploration costs and accelerate the learning process of the centralized learner [80]. Standard DRL stability enhancements, such as separating policy and target net-

works (as seen in DQN and its variants) and using experience replay buffers to break the correlation between sequential training samples, remain critical components in these frameworks [78, 80, 139].

2.4.4 Challenges and Future Research Directions

Despite the promising theoretical results and demonstrable performance improvements in simulation environments, several critical research gaps and challenges persist in the field of RL-driven service placement. A major and recurring limitation highlighted across numerous studies is the reliance on simulation-based evaluations (often using tools like iFogSim, MATLAB, or bespoke Python environments) rather than comprehensive validation in real-world, large-scale operational deployments [10, 44, 48, 151, 95]. This practice raises concerns about the practical applicability, robustness, and scalability of proposed DRL solutions in heterogeneous and dynamic environments [10, 44, 151]. Compounding this issue is the lack of standardized evaluation metrics across different studies, making meaningful comparative benchmarking of various RL techniques extremely difficult [10, 151, 95]. Future work should also prioritize enhancing the scalability of AI models to coordinate seamless operation across highly distributed and heterogeneous fog/edge nodes [10, 44, 80]. Other prospective research directions include exploring multi-agent reinforcement learning (MARL) for cooperative or competitive optimization [78, 130, 139], incorporating energy efficiency as a key objective function constraint [78, 130, 80], and adapting solutions to manage dynamically changing network topologies and node mobility [78, 139, 80].

2.5 Conclusions

Chapter 2 provided an overview of the foundational concepts and state of the art underlying service deployment in the Cloud–Edge Continuum. We first introduced the architectural and operational characteristics of the Continuum, emphasizing its heterogeneous nature and the challenges posed by latency, scalability, and security requirements. We then surveyed existing approaches to service placement, highlighting application-driven requirements, optimization models, and deployment strategies adopted in cloud, fog,

and edge environments. This analysis revealed persistent gaps, particularly in handling multi-operator scenarios, incorporating security constraints, and supporting dynamic, large-scale deployments.

The chapter also examined workflow description languages, tracing the evolution from centralized orchestration models to more expressive and decentralized choreography-based solutions. Despite significant progress, current languages still fall short in supporting adaptive, QoS-aware, and security-aware workflows suitable for highly heterogeneous infrastructures. Finally, we reviewed reinforcement learning techniques for service placement, which offer promising results in addressing the complexity and dynamics of the Continuum. However, existing RL-based solutions rely heavily on simulators, lack standardized evaluation metrics, and often overlook critical objectives such as security and vulnerability-aware optimization.

Overall, this chapter established the conceptual and technological background motivating the methodologies developed in the rest of the thesis. The identified research gaps, particularly in QoS-aware deployment, vulnerability-aware placement, workflow adaptivity, and intelligent optimization, directly inform the frameworks and models presented in the following chapters.

The following chapter (Chapter 3) introduces the core framework used in this thesis. The proposed framework abstracts from any underlying deployment technology and identifies all the components involved in the deployment process.

3

QoS-Aware Deployment Framework

Building upon the solutions and gaps in the state-of-the-art in service deployment and service placement highlighted in Chapter 2, this chapter presents a new framework for QoS-aware service deployment in the Cloud-Edge Continuum. The framework introduces a graph-based methodology that models both service compositions and deployment facilities, enabling systematic matching of requirements to capabilities. It generates deployment recipes that preserve non-functional properties such as latency, confidentiality, and integrity, thereby extending current deployment practices toward property-driven and technology-agnostic solutions. The content of this chapter is based on the following publication: *M. Anisetti, F. Berto, and R. Bondaruc, "QoS-Aware Deployment of Service Compositions in 5G-Empowered Edge-Cloud Continuum", in 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), lug. 2023, pp. 471–478.*

3.1 Introduction

The increasing adoption of cloud-native technologies and edge computing has radically changed how applications are developed and deployed. While containers and virtual machines enable modular and scalable applications, the rise of latency-sensitive and data-intensive services has exposed the limitations of purely cloud-based deployments. Edge computing offers proximity to data sources and users, improving responsiveness, but lacks the com-

putational power and elasticity of the cloud. The Cloud-Edge Continuum emerges as a paradigm that unifies these environments, promising deployments that can adapt to heterogeneous requirements while exploiting the strengths of each layer.

Despite this potential, current deployment approaches remain limited. Most existing solutions focus on resource allocation in cloud or serverless environments, but they do not support complex workflows that require guarantees on non-functional properties such as latency, confidentiality, or integrity. Furthermore, the heterogeneity of infrastructures—spanning cloud, telco edge, and on-premises facilities—poses challenges for technology-agnostic deployment. Developers are often forced to manually adapt applications or accept suboptimal placements, undermining the promise of seamless continuum computing.

The goal of this chapter is to introduce a framework for QoS-aware deployment of service compositions in the Cloud-Edge Continuum. The framework provides an abstract, graph-based representation of both service workflows and continuum facilities, and defines a matching methodology that associates services with suitable resources while satisfying requirements and constraints. From this mapping, the framework generates deployment recipes, which are executable plans tailored to the available facilities.

The main contributions of this chapter are: i) a reference architecture for continuum deployment that supports heterogeneity and property-driven decisions, ii) a methodology based on annotated graphs and matching functions for mapping workflows to facilities, and iii) an experimental evaluation demonstrating the feasibility and efficiency of the approach in representative scenarios.

3.1.1 Chapter Outline

The remainder of this chapter is organized as follows. Section 3.2 introduces a new notion of Edge-Cloud Continuum involving both 5G telco Edge node and on-premises Edge node. Section 3.3 presents a novel methodology for QoS-aware deployment of composed services in the Edge-Cloud Continuum and a suitable Architecture implementing the methodology and generating executable recipes for the deployment. Section 3.4 presents a preliminary

experimental evaluation showing the feasibility of our methodology. Finally, Section 3.5 presents the chapter conclusions.

3.2 Scenario, Requirements and Architecture

Our reference scenario considers i) a client that wants to deploy its workflow of services s_i specifying QoS requirements and constraints to be satisfied on the Edge-Cloud Continuum; ii) Cloud Service Providers (CSPs) offering deployment facilities f_i for third-party services on the Edge-Cloud Continuum; iii) facilities offering specific capabilities c_i to satisfy QoS requirements and constraints. In this work we consider an advanced Edge-Cloud Continuum where Edge nodes can be: i) telco nodes (i.e., 5G Multi-access Edge Computing - MEC) based on an agreement between the CSP and a given telco operator offering their core network capabilities to be part of the CSPs Continuum (e.g., AWS Wavelength); ii) on-premises nodes based on the deployment facilities on the client's premises enabled by the CSP for services deployment (e.g., using AWS Greengrass). When the client asks to deploy a given workflow of services, the CSP matches the service workflow, QoS and constraints with the capabilities and peculiarities of its Edge-Cloud Continuum deployment facilities associated to the specific client in order to find all the feasible deployment configurations. Among them the CSP selects the configuration to be deployed according to internal policies, such as considering Cloud as the preferred service deployment due to lower operating costs. For instance, in case of two candidate configurations, one using 5G and Cloud facilities and another using on-premises and Cloud facilities, the latter would be selected. The CSP then generates the deployment recipe for the selected configuration and executes the deployment on the continuum.

Example 3.1 shows the scenario used in the rest of the chapter to present our methodology.

Example 3.1 (The Scenario)**.** *Let us assume that a client wants to deploy a machine learning workflow made of 4 sequential services: s_1 gathering data, s_2 normalizing data, s_3 generating a model out of the collected data (e.g., training a decision tree), and s_4 saving the model for further usage (e.g., for prediction). Let us assume that the client requires that data accessed by s_1 should be pro-*

tected for confidentiality and the model generated by s_4 should be protected from tampering by controlling its integrity. Let us assume that the client expresses a constraint on the communication links between s_1 and s_2 and between s_2 and s_3, which must carry large volumes of raw data, requiring a bandwidth of at least 200 Mbit/s. Let us also assume the client has a contract with the CSP for Continuum facilities including on-premises machine f_1, 5G Edge node f_2 in the proximity of its premises, and Cloud node f_3. The CSP ensures some capabilities (c) on the facilities and the network links connecting them. In particular, f_1 ensures confidentiality at rest via isolation of the storage from the direct control of CSP (c_1), and f_3 provides a service (s_I) ensuring data integrity at rest (c_2). In addition, all the internal links, that is the links between two services deployed on the same facility, have infinite bandwidth (c_3), while the 5G link between f_1 and f_2 provide at least a 500 Mbit/s bandwidth (c_4).

3.2.1 Edge-continuum deployment Requirements

In order to support the Scenario in Section 3.2, the CSP should be empowered with an advanced deployment architecture addressing the following requirements:

- **(R1) Continuum-readiness**: it should seamlessly deploy services on all the different Continuum premises.
- **(R2) Property-driven**: it should be driven by QoS properties and constrains expressed by the client.
- **(R3) Technology agnostic**: it should be capable to handle heterogeneous deployment facilities regardless the underlying virtualization technology.
- **(R4) Comprehensive model**: it should provide a way as general as possible to represent pipelines and facilities, without limiting the choice in topology and data flow.
- **(R5) Interoperability**: it should be able to interact with CSP facilities through software hooks (e.g., APIs enabling service deployment and monitoring).

Table 3.1: Related work comparison.

Author	Ref.	R1	R2	R3	R4	R5	R6
K. Fu et al.	[74]	✓			✓		∼
A. Orive et al.	[131]	✓	∼		✓	✓	
A. Brogi et al.	[35]		∼	✓	✓		
V. Casola et al.	[40]	✓	✓	✓		✓	
S. Nastic et al.	[122]	✓		✓			
N. Akhtar et al.	[6]		∼	✓		✓	∼
A. Das et al.	[52]			✓	✓	✓	
M. Anisetti et al.	[16]		✓	✓			
J. Quenum et al.	[143]		✓	✓		✓	
Our Work		✓	✓	✓	✓	✓	✓

- **(R6) Context adaptability**: it should perform deployment life-cycle management by re-deploying services when changes in the environment occur.

To the best of our knowledge, no deployment architecture is currently capable to fully address the above requirements offering a comprehensive and adaptable approach to permanently deploy services in the Continuum taking into account client-defined advanced properties and constrains. Few solutions exist for application deployment, which are compared in Table 3.1 with respect to the above requirements (full and partial support of a requirement is denoted with ✓and ∼ respectively). Some considered works ([74, 131, 40, 122]) address the Edge-Cloud Continuum scenario, but only the architecture presented in [40] can seamlessly deploy applications along the Continuum since it is fully independent of the specific technology and CSP involved. QoS requirements and constraints are taken in account, even partially, by most of

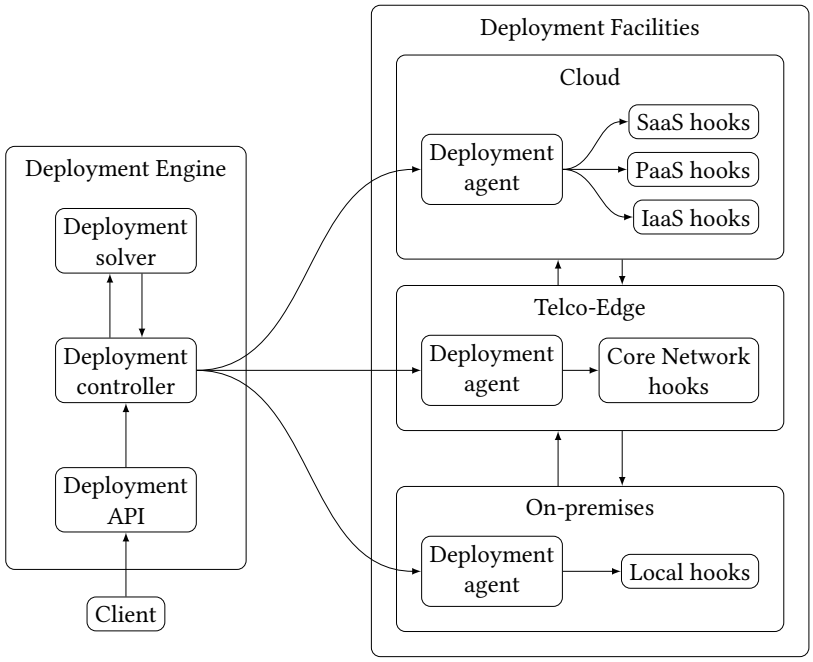

Figure 3.1: Deployment Architecture for Edge-Cloud Continuum.

the works ([131, 35, 40, 6, 16, 143]), but only [131, 35] provide a comprehensive way to model both applications and environment. Finally, there is no solution that performs a life-cycle management of the deployment, adapting to changes in context. In short, all solutions have drawbacks, whether they be the limited application scenario, the dependence on specific technologies, or the inadequate composition model. Our architecture, instead, addresses all the aforementioned requirements, providing a QoS-driven deployment system for the Continuum.

3.2.2 Deployment Architecture

To support the scenario in Section 3.2 and ensure the requirements in Section 3.2.1, we propose a deployment architecture capable of guaranteeing a seamless QoS and constraints preserving execution of a given service work-

flow in the Continuum.

Figure 3.1 schematizes our system architecture made of i) a *Deployment Engine* service that targets the deployment facilities through *Deployment Agents*, and ii) a set of *Deployment Facilities* of different nature, including Cloud, telco-Edge and on-premises nodes (R1). The client interfaces with the deployment engine through *Deployment API* providing the service workflow to be deployed and the QoS/constraints specification in a machine-readable format [(R2) and (R4)]. The *Deployer Solver* chooses the most appropriate deployment configuration interacting with the *Deployment Controller* which is in charge of i) interrogating Deployment Facilities on their capabilities and ii) delivering the deployment recipes, using the *Deployment Agents*.

The Deployment Agents are responsible to build the deployment continuum across the facilities by driving the service deployment (R3). In order to support QoS and constraint-aware integration, the CSP exposes to the Deployment Agents suitable hooks to relevant resource (e.g., resource manager for deployment) or services (e.g., services offering security features) constituting their capabilities (R5). For instance, the CSP can offer a hook to access non-functional certificates (i.e., using certification scheme in [11, 13]) proving some capabilities or to invoke authentication services to support authentication requirements.

We note that in case of necessity (e.g., changes in the QoS or budget constraints) the given workflow can be re-deployed via re-executing the deployment matching with modified QoS and constraints (R6). The re-deployment can be also used to handle service migrations and in general changes occurring post-deployment.

In the following we describe the peculiarities of the Continuum *Deployment Facilities* in terms of architecture and capabilities.

3.2.3 Cloud

Cloud solutions offer to their users managed services and infrastructure with scalable amount of resources. This allows users to deploy and integrate their services and products, relieving them from most management tasks. Generally, the control plane of the system is handled by a cloud provider, managing the services life-cycle and ensuring their availability. The most common type

of deployment used in this context are based on containerized services or on virtualized hosts.

The Cloud is the most flexible and powerful facility in the Continuum in terms of offered capabilities. It supports non-functional QoS via PaaS or IaaS services enabling them for the deployed services. It also partially supports constraints, for instance via resource scalability, network latency, high bandwidth and connectivity enabled by virtualized network and local and distributed data centers. However, this support is normally very limited due to the public internet connectivity and the shared environment used in standard Cloud configurations. The Cloud currently suffers from the lack of transparency impacting non-functional properties such as privacy [18].

To support our deployment architecture, the Cloud facility has to provide hooks for i) resource management offering deployment and configuration of containerized and/or virtualized services; ii) configurations of services relevant to support given non-functional properties; and iii) workflow-level networking management.

3.2.4 Telco-Edge

Telco-Edge is an innovative Edge scenario enabled by 5G via integrating mobile networking capabilities. The state-of-the-art solution for automated service management in such systems is based on MEC [65, 77, 102, 94]. MEC allows integration of container and VM based services with the 5G core network, mutually exposing their functionality. Differently from traditional Cloud facilities, i) the edge nodes are connected to 5G radio antennas, allowing fast communication to and from mobile devices; ii) the service provisioning to mobile devices is backed by unique device and user identification through IMEI and SIM identifiers, supporting strong authentication; iii) the telco edge nodes can be geographically closer to their users, enabling new notion of data privacy by proximity, similarly to the notion of on-premise privacy; and iv) the 5G standard allows users to allocate virtual network slices, ensuring bandwidth availability and network performance levels and latency.

In addition, by adopting the telco-Edge facilities i) the bandwidth between the edge node and other network nodes is allocable; ii) the in-motion

data can be contained within the boundary of the telco network; and iii) additional services (i.e., identification, network monitoring, authentication) for the users connected through mobile network are available.

To support our deployment solution, the telco-Edge facility have to offer hooks on resource management and service configuration similarly to the Cloud but also additional peculiar hooks for i) network-level user management and authentication, and ii) network resource allocation (5G network slices) for bandwidth and latency management via MEC.

3.2.5 On-premises

With on-premises we consider deployment facilities that are fully under the control of the owner, but are equipped with CSP services to make them part of the Continuum. They can be realized via VMs/containers or physical machines. Such facilities have normally stronger resource limitations compared to Cloud or Edge environments, lacking scalability and elasticity or not providing specific hardware. On the contrary, they i) have strong properties of high confidentiality and privacy, for instance ensuring that data cannot physically leave the organization; ii) have the lowest latency to the devices within the organization; and iii) leave the owners full control of the execution environment.

In order to be part of the Continuum, on premises facilities should i) allow our Deployment Agents to be executed in a supported deployment environment; ii) allow access to resources to arrange the service deployment; and iii) offer hooks to handle resources for local deployment of services, connectivity to the continuum and access to the relevant local services if needed. Normally, on premises hooks have very restricted access to local services, data and deployment resources making their use in the continuum very challenging.

3.3 Methodology

Figure 3.2 shows our methodology for the Edge-Cloud Continuum scenario in Section 3.2 based on our architecture in Figure 3.1. The client defines the service composition workflow to be deployed and annotates it with QoS re-

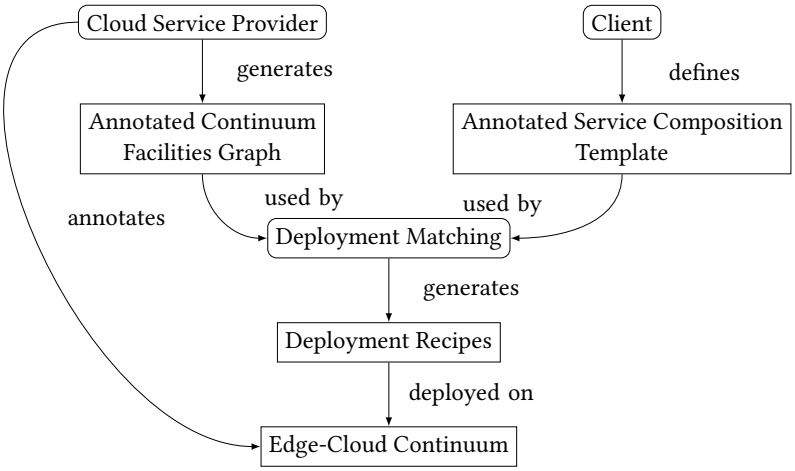

Figure 3.2: Our Methodology.

quirements and constraints forming an *Annotated Service Composition Template*. The service provider annotates its facilities with the relevant properties, indicates hooks reachable by Deployment Agents, and produces the *Annotated Deployment Facilities Graph*. The client submits the request for deployment by submitting the Service Composition Template via Deployment API to our Deployment Engine. The Service Composition Template and the Annotated Deployment Facilities Graph trigger the *Deployment Matching* process executed by our Deployer Solver in order to generate the *Deployment Recipes* to be used to deploy the given workflow of services on the Edge-Cloud Continuum.

3.3.1 Annotated Service Composition Template

A Service Composition Template is an abstract representation of the workflow of services that a client wants to deploy. It describes the services, their execution parameters and their interconnectivity configuration.

Definition 3.1 (Service Composition Template)**.** *A Service Composition Template is a directed graph* $T = (S, E)$ *where* $s_i \in S$ *are services constituting the graph vertexes, and* $e_i \in E$ *are graph edges modeling the interactions between*

two services on both control and data plane. We note that, since $T = (S, E)$ is a directed graph, each edge represents a one-way interaction, while a two-way communication requires a cycle made by two edges.

The Service Composition Template can be annotated with specific non-functional requirements $r_i \in R$ and constraints $k_i \in K$ on resources. Following the notation in [15], a requirement r_i is a pair $(\hat{r}, Attr)$, where $r_i.\hat{r}$ is an abstract requirement from a shared vocabulary of properties (e.g., confidentiality, integrity) and $r_i.Attr$ is a set of attributes specifying the low-level characteristics that should be provided. The attribute values induce a hierarchy H_R of requirements $(R, \preceq_R)$, where R is the set of requirements and $\preceq_R$ is the partial order. Similarly, a constraint $k_i \in K$ is a tuple $(\hat{k}, Val, Op)$, where $k_i.\hat{k}$ is a resource (e.g., bandwidth, on-premises), $k_i.Val$ is the desired value and $k_i.Op$ is the operation on that value. The resource determines what type of value can be specified, such as integers, booleans, lists, and what operations can be applied (e.g., $=, <, \geq, \in$).

Definition 3.2 (Annotated Service Composition Template)**.** *Let $T = (S, E)$ be a Service Composition Template. The Annotated Service Composition Template $T^{R,K}$ is generated annotating vertexes and edges in the template $T = (S, E)$ with non-functional requirements $r_i \in R$ and constraints $k_i \in K$.*

For instance, a security requirement $r_i = (Confidentiality, AES256)$ can be associated to an edge (i.e., communication channel) e_i, denoted as $e_i^{r_i}$, while a constraint $k_j = (On\text{-}premises, \texttt{true}, =)$ can be associated to a vertex (i.e., a service of the workflow) s_j denoted as $s_j^{k_j}$ indicating that the deployment must be performed on an on-premises facility.

Example 3.2 (Annotated Service Composition Template)**.** *Considering Example 3.1, we can represent the client service workflow as the annotated template $T^{R,K}$ with the set of services $S = [s_1^{r_1}, s_2, s_3, s_4^{r_2}]$ and the set of edges $E = [e_1^{k_1}, e_2^{k_1}, e_3]$. The expressed requirements are r_1 = (Confidentiality, Isolation) and r_2 = (Integrity, Rest), while the posed constraint is k_1 = (Bandwidth, 200, $\geq$).*

3.3.2 Annotated Continuum Facilities Graph

The *Cloud Service Provider* (CSP) models its Continuum facilities for a given client as a Continuum Facilities Graph.

Definition 3.3 (Continuum Facilities Graph)**.** *The Continuum Facilities Graph is a directed graph $G = (F, L)$ made of a vertex $f_i \in F$ for each facility provided by the CSP. L is the set of edges (here called links) l_i connecting the vertices of the graph. Two vertices f_i and f_j can be connected by a link l_i if facility f_j is reachable from facility f_i either through the open Internet or a dedicated private channel.*

Note that the Continuum Facilities Graph is directed and therefore each link models a one-way connection. We can however express both two-way and intra-node communication capabilities by defining cycles. In particular, intra-node communication is represented as a cycle of length 1, i.e., a loop.

The Continuum Facilities Graph generated is then annotated with non-functional capabilities $c_i \in C$. Capabilities represent a mechanism to support both non-functional properties and constraints such as confidentiality, latency, performance to name but a few. A capability is defined as a tuple $(\hat{c}, Spec, Op, Impl)$, where $c_i.\hat{c}$ is either a property or a resource, $c_i.Spec$ is an attribute if $c_i.\hat{c}$ is a property, a value otherwise, $c_i.Op$ is an operation on $c_i.Spec$ (if it is an attribute, Val is always a $=$), and $c_i.Impl$ is a set, even empty, of key-value pairs describing how the facility implements the capability. The annotated data life-cycle is managed by the service providers, possibly using certification-based solutions. The selection and implementation of the associated methodology is out of the scope of this chapter.

Definition 3.4 (Annotated Continuum Facilities Graph)**.** *Let $G = (F, L)$ be a Continuum Facilities Graph. The Annotated Continuum Facilities Graph G^C is generated annotating vertexes and edges in the Continuum Facilities Graph $G = (F, L)$ with the capabilities $c_i \in C$ (e.g., latency, bandwidth, resources).*

For instance, a security capability $c_i = (Channel\_encryption, AES256, =)$ can be associated to a link l_i denoted as $l_i^{c_i}$, while a capability $c_j = (Edge, \texttt{true}, =)$ can be associated to a vertex (i.e., a facility of the SP) f_j denoted as $f_j^{r_j}$.

Together, Annotated Service Composition Template and Annotated Continuum Facilities Graph provide a general framework to describe most kind of pipeline topologies in the Continuum, addressing (R4).

Example 3.3 (Annotated Continuum Facilities Graph)**.** *Considering Example 3.1, we can represent the CSP facilities as the annotated graph* G^C *with the set of facilities* $F = [f_1^{c_1}, f_2, f_3^{c_2}]$ *and the set of links* $L = [l_1^{c_3}, l_2^{c_4}, l_3^{c_3}, l_4, l_5, l_6, l_7^{c_3}]$*. The provided capabilities are* c_1 *= (Confidentiality, Isolation, =, []),* c_2 *= (Integrity, Rest, =, [service: sI; mode: interception]),* c_3 *= (Bandwidth,* $+\infty$*, =, [])* *and* c_4 *= (Bandwidth, 500,* $\geq$*, []).*

3.3.3 Deployment matching

The deployment matching process searches for the most suitable solution for the QoS-aware deployment of a given service workflow on the Continuum. It takes as input the Annotated Continuum Facilities Graph G^C and the Annotated Service Composition Template $T^{R,K}$, generates a set of suitable deployment solutions M and among them finds the one $\hat{M}$ that better satisfies a given CSP policy (e.g., the lowest operational cost).

The set of suitable deployment solutions M is defined on $(S \in T^{R,K}) \times (F \in G^C)$ so that: i) for every edge $e_i \in E$ between any two vertices s_i and $s_j \in S$ there is a link $l_i \in L$ between the matching vertices f_i and $f_j \in F$; ii) for every pair (s_i, f_i) in M, the capabilities c_i of f_i satisfy the requirements r_i and the constraints k_i of s_i; iii) for every edge e_i between any two vertices s_i and $s_j \in S$, the capabilities c_i of l_i between the matching vertices f_i and $f_j \in F$ satisfy the requirements r_i and the constraints k_i of e_i. If the set of suitable deployment solutions M is empty, it means that the deployment of the service workflow cannot take place in the given continuum, given the QoS requirements and constraints. If the set of suitable deployment solutions M is not empty, the deployment matching orders them according to the CSP policy and selects the first one in the order.

Given the set of deployment solutions M, if the CSP policy refers to operational cost reduction only, a solution like the one in [16] can be adopted. We will investigate the impact of more articulated CSP policies as well as the adoption of an optimization approach for finding the suitable deployment solution and contemporaneously addressing such CSP policy in the following

Algorithm 1 Pseudocode of our deployment matching exhaustive algorithm.

```
procedure MATCHING(S, F)
    matches = ∅
    ▷ Iterate services permutations
    for service_perm in perm(S) do
        ▷ Services partitioning in |F| facilities
        for part in partitions(service_perm, |F|) do
            ▷ Test if all requirements are met
            valid ← true
            for r_i in R do
                if r_i(S, F, part) == false then
                    valid ← false
                    break
                end if
            end for
            if valid then
                ▷ Match found
                matches = matches ∪ {part}
            end if
        end for
    end for
    return matches
end procedure
```

chapters.

Algorithm 1 shows the pseudocode of our matching function. The algorithm i) iterates over all the permutations of services altering the elements starting from the beginning of the list; ii) for each permutation it iterates on its partitions starting from the beginning of the list, thus leaving the largest partitions containing the most preferred facilities; iii) for each permutation it checks whether all requirements are met, if that is the case it adds it to the results partition, otherwise it breaks to the inner for loop; iv) finally it returns the set of results.

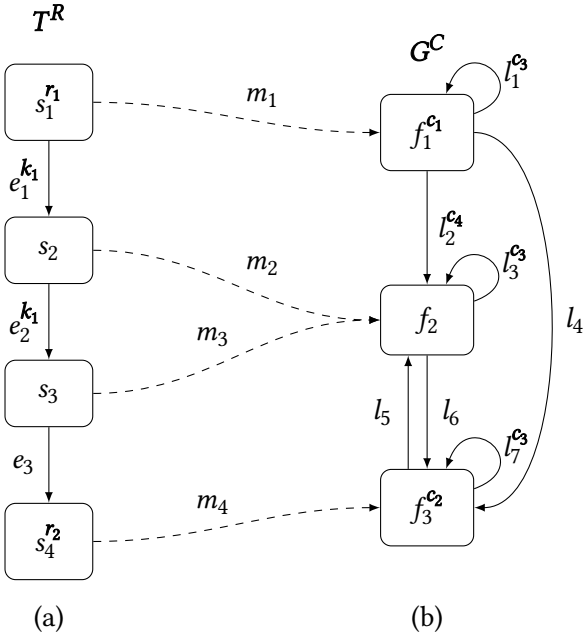

Figure 3.3: Annotated Deployment Graphs including (a) Annotated Service Composition Template and (b) Annotated Continuum Facilities Graph, with the resulting matching.

Example 3.4 (Deployment matching)**.** *Let us consider the scenario in Example 3.1, the Annotated Template $T^{R,K}$ and the Annotated Graph G^C, in Example 3.2 and Example 3.3 respectively. Let us now apply our matching function in Algorithm 1 to the services in $T^{R,K}$ and the facilities in G^C retrieving a set of possible matching M. Figure 3.3 shows one of the possible solution in $M = \{m_1, m_2, m_3, m_4\}$, where $m_1 = (s_1, f_1)$, $m_2 = (s_2, f_2)$, $m_3 = (s_3, f_2)$, and $m_4 = (s_4, f_3)$. Here, s_1 is matched with f_1 since it is the only facility ensuring confidentiality. Similarly, s_4 is matched with f_3 since it is the only facility providing a way to check integrity. Lastly, s_2 and s_3 are matched with f_2 to provide large bandwidth between the first three services.*

3.3.4 Deployment Recipes

The purpose of the above deployment matching is to assign each service to a facility in such a way that all requirements (i.e., QoS and constraints) can be fulfilled and the CSP internal policy satisfied. However, a mere assignment is not always enough to enforce the desired properties. For instance, low latency can be achieved by simply assigning services to the physically closest facility, while, on the contrary, communication channel confidentiality requires configuring a component or service to handle message encryption and decryption. To address the above deployment challenges, our methodology enriches traditional deployment recipes with hooks metadata. This metadata is consumed by our Deployment Agents to enable relevant properties configuring or embedding facility's services in the service workflow to be deployed.

Recipes are generated for each service of the workflow in $\hat{M}$ and contain the operational instructions for the deployment of both the service and the supporting facility's components/services. In particular, a recipe consists of three parts: i) the deployment configuration of the service, including the image to be used and resources to be allocated; ii) a description of the support components/services to be integrated and their parameters (e.g., for an authentication component it includes the list of user credentials); iii) modality of integration (i.e., none, interception or wrapper).

Example 3.5 (Deployment Recipe)**.** *Considering the selected matching $\hat{M}$ in Example 3.4, according to our methodology, the relative deployment recipes are generated for all the services $s_i \in \hat{M}$. For simplicity let us focus on s_4 recipe only. Figure 3.4 shows an excerpt of the s_4 recipe provided to the deployment agent in f_3, along with the final deployment graph for $\hat{M}$. According to the recipe, the agent, in order to guarantee integrity, has to deploy an additional facility's service (service: s_I) that intercepts (mode: interception) incoming data from s_3, computes and adds to data a cryptographic checksum, and delivers it to s_4 for storing.*

We note that details on how the recipes are generated out of the given selected matching $\hat{M}$ is out of the scope for this work. We will further investigate this topic in our future works.

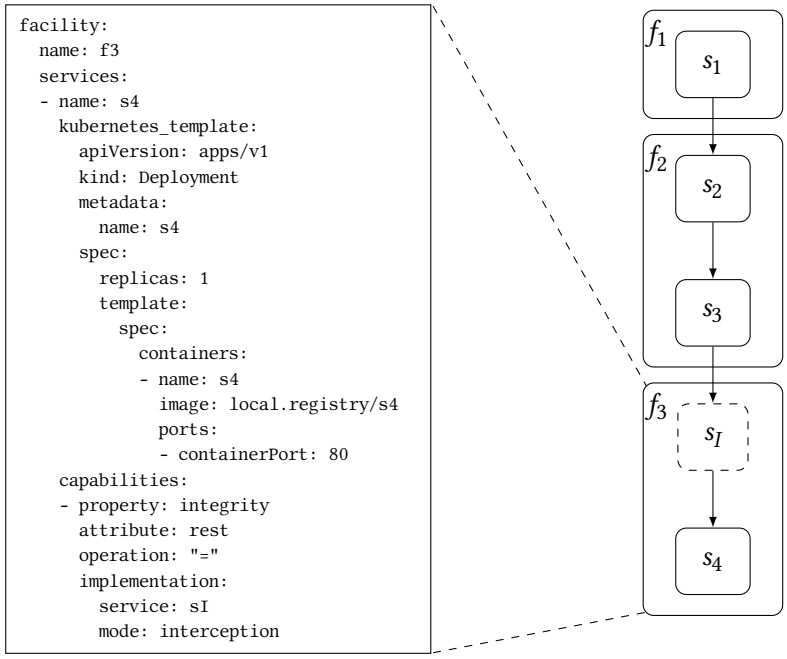

Figure 3.4: Deployment Recipe for facility f_3.

3.4 Experimental Evaluation

In this section we first describe our experimental settings realizing the scenario in Section 3.2 and then present a preliminary performance evaluation.

3.4.1 Experimental setup

We realized the Edge-Cloud Continuum of our scenario in Section 3.2 as follows. The software stack used in the Cloud facility was based on MicroK8s. It allowed to simulate cloud-hosted VPS nodes in a cluster formation supporting auto-scaling as we expect by a Cloud provider. The telco-Edge facil-

ity was implemented based on Open Source MANO (OSM)[1], Free5GC[2] and MicroK8s[3], respectively implementing the MEC, a simulated 5G core network, and the resource manager. OSM was designed to integrate with the core network, exposing the services hosted in Kubernetes to the users and other nodes as Network Functions. The on-premises facility was based on MicroK8s, configured to handle limited resources. All the nodes were hosted in the same data center, and were equipped with 4 virtual cores clocked at 2.09GHz and 32 GB of RAM and running Linux 5.4.0 (Ubuntu 20.04 LTS). All the nodes of the continuum were empowered by our deployment agents communicating with our Deployment Engine using the same virtualized network used by the continuum node.

Our Deployment Engine was implemented using Python 3.10.9 and has been executed on a machine running Linux 5.15.102 (NixOS) equipped with an AMD 5900x and 32 GB of RAM.

3.4.2 Performance evaluation

The computational effort needed to deploy a given service workflow using our Deployment Engine is clearly dominated by the effort required by our matching function that can be estimated as $\mathcal{O}(|F|^{|S|})$ where $|F|$ and $|S|$ are the cardinality of Continuum facilities and services in the workflow respectively.

Figure 3.5 shows the execution time requested by the matching function varying the number of services $s_i \in S$ in the workflow and the number of facilities $f_j \in F$ of the continuum using logarithmic scale. The performances were computed on the average of 5 executions for each combination of services and facilities. For each execution we randomly generated facilities' capabilities and services' requirements. As expected the number of services $|S|$ dominates the exponential growth of execution time.

The matching algorithm terminates in less than one second when evaluating the cases of 6 services in 5 facilities and 8 services in 3 facilities. As the number of services increases, the time of execution grows rapidly: the execution time with 12 services and 3 facilities is 8.11s, which is the last experi-

[1] `https://osm.etsi.org/`
[2] `https://www.free5gc.org`
[3] `https://microk8s.io`

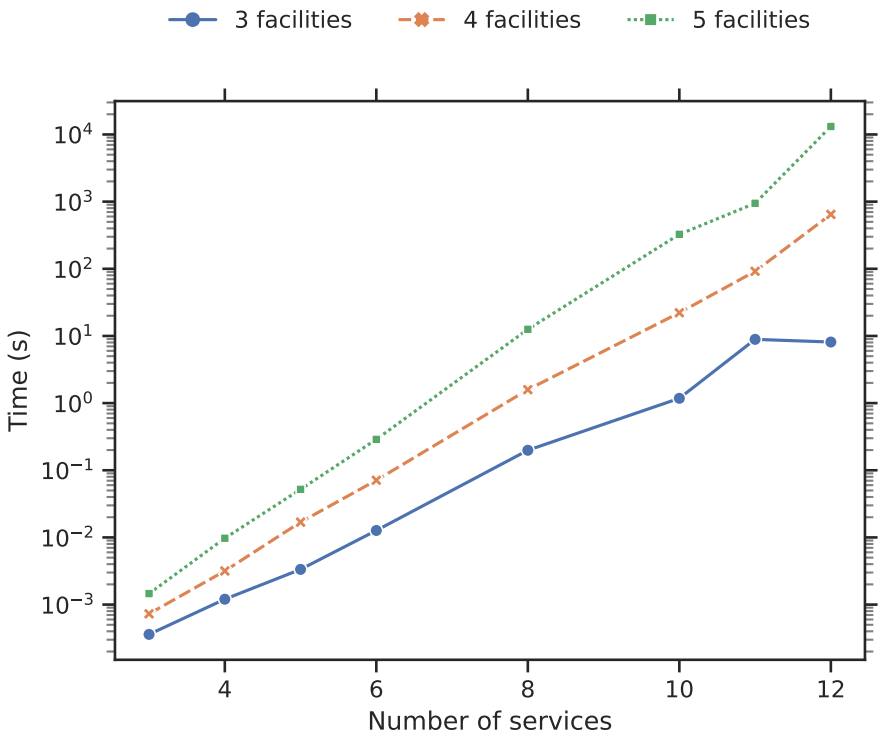

Figure 3.5: Execution time with increasing number of services s_j in the workflow and considering 3, 4 and 5 facilities f_j.

ment configuration to terminate under 10s. The cases 8 services in 5 facilities and 10 services in 4 facilities terminate respectively in 12.6s and 22.1s. With larger numbers of facilities and services the algorithm becomes too slow for practical use: the configuration 11 services in 4 facilities terminates in 91.7s. The last configuration shown in Figure 3.5 considers 12 services in 5 facilities and has terminated in 13143s.

3.5 Chapter Conclusions

In this chapter we described a novel methodology for deploying composed services in Edge-Cloud Continuum focused on guaranteeing advanced QoS requirements. The main idea is to exploit the capabilities and the peculiarities of the different continuum landing platforms to ensure QoS at deployment time. Our deployment methodology is based on a machine-readable description of the service composition annotated with the QoS requirements and a meta-description of the capabilities of the landing platforms involved in the continuum. Based on these machine-readable descriptions, a set of feasible deployment solution are generated and the relative deployment recipes for the service composition are selected according to a given policy. We show the feasibility of our solution in our preliminary experimental evaluation.

The following chapter (Chapter 4) extends the presented methodology to include vulnerability assessment in the placement decision process. It also provides a strategy to mitigate recursive migrations, the primary cost driver in the Continuum environment, and to optimize resource allocation according to the CSP policy.

4

Vulnerability-Aware Deployment and Migration Strategies

This chapter presents a vulnerability-aware approach to service deployment in the Cloud-Edge Continuum. The proposed methodology integrates vulnerability assessment directly into the methodology presented in Chapter 3, accounting for both migration costs and resource consumption. It enables proactive mitigation of recursive migrations and supports secure, cost-efficient deployment strategies. Collectively, Chapter 3 and Chapter 4 contribute to the foundation of a QoS- and security-aware approach for deploying services across Cloud and Edge. The content of this chapter is based on the following publication: *R. Bondaruc, N. Schnepf, R. Badonnel, C. A. Ardagna, and M. Anisetti, "Vulnerability-Aware Secure Service Deployment in Cloud-Edge Continuum", IEEE Transactions on Network and Service Management, pp. 1–1, 2025.*

4.1 Introduction

The Edge–Cloud Continuum integrates Edge Computing, which brings computation closer to data sources, with traditional Cloud infrastructures. This seamless integration supports real-time applications through low latency [135] and enables flexible service placements to satisfy diverse QoS

requirements [14]. As distributed systems proliferate, non-functional properties such as performance, reliability, and security become critical alongside functional correctness. In particular, data integrity is essential in domains like financial transactions, legal records, and critical infrastructures where unauthorized data alterations can cause severe disruption.

To meet these needs, deployment must allow explicit specification of user requirements and provider capabilities. This two-way negotiation enables services to exploit the Continuum's advantages without altering application logic. However, the shared and dynamic nature of this environment complicates security: services migrate between facilities, each introducing vulnerabilities that reshape the host's security footprint. Without visibility into these risks, safe deployment—especially in low-risk domains such as power grids or transport networks—becomes impossible.

We therefore argue for proactive vulnerability assessment prior to deployment, enabling users to specify required security levels and providers to adjust placements dynamically as threats evolve. This is particularly important in the Continuum's multi-tenant setting, where colocated services (from different tenants or the CSP itself) can impact one another's QoS and security guarantees. Maintaining a global, up-to-date view of the environment is thus vital.

Our proposed methodology addresses these challenges by: i) modeling vulnerabilities as first-class requirements in deployment, ii) defining a vulnerability scoring mechanism to quantify each facility's security posture, iii) introducing a cost-aware deployment strategy that respects both security and QoS constraints, and iv) providing a framework for adaptive migration, ensuring requirements remain valid throughout a service's lifecycle.

The model is illustrated in Figure 4.1, where a client deploys a workflow of services s_i with QoS requirements r_i, the CSP offers facilities f_i with capabilities c_i, and workflows are matched to facilities, with monitoring and migration triggered when capability changes invalidate requirements.

This approach embeds security directly into deployment logic, enhancing resilience, reducing migration overhead, and optimizing the alignment between service needs and infrastructure capabilities across the Edge–Cloud Continuum.

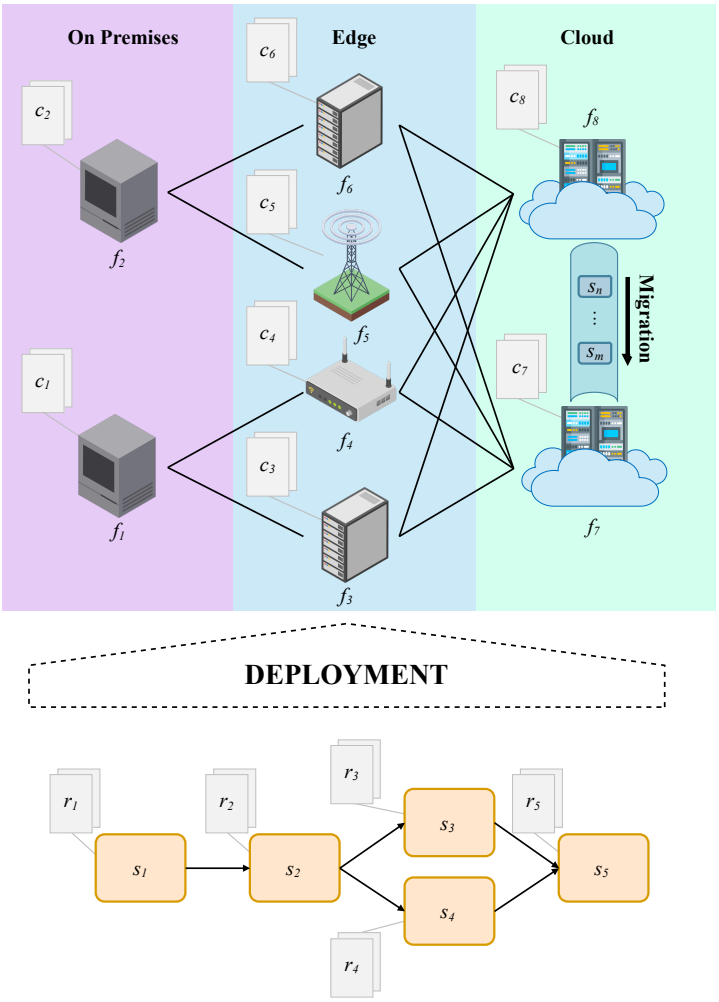

Figure 4.1: Workflow of services deployed in the Edge-Cloud Continuum.

4.1.1 Chapter Outline

The remainder of the chapter is organized as follows. In Section 4.2 the related works are reviewed. In Section 4.3 we propose the extended methodology for service deployment in the Edge-Cloud Continuum. In Section 4.4,

we provide the model formalization. In Section 4.5 we describe the vulnerability driven placement optimizing costs relevant to CSPs. In Section 4.6 we put forward a strategy for service migration. In Section 4.7 we evaluate our methodology showing its effectiveness. Finally, the conclusions are addressed in Section 4.8.

4.2 Related Work

The field of software vulnerability assessment and management, as well as the placement of services in distributed computing environments, has seen extensive research over the years. This section reviews key contributions in these domains, highlighting existing methodologies and identifying open challenges that motivate the work presented in this chapter.

4.2.1 Vulnerability Assessment

Software vulnerabilities have been widely studied, and vulnerability management is generally divided into two main tasks: assessing and remediating vulnerabilities [72]. Considerable research has focused on supporting the assessment process through techniques for discovering unknown vulnerabilities, specifying them once discovered, and detecting their presence using description databases [27]. These efforts are complemented by approaches aimed at automating or streamlining the remediation of identified vulnerabilities.

Discovering vulnerabilities is a fundamental activity that gives purpose to the vulnerability management process. Without identified vulnerabilities, the process would be meaningless. Therefore, the capability to uncover new threats within the environment is essential. With that respect, testing methods, including fuzzing approaches, provide a powerful strategy to identify unknown vulnerabilities. Software applications are commonly designed with a set of specific goals in mind in order to provide effective solutions to the stated requirements. While developers pursue efficient functional constructions, testers perform tasks for identifying correctness, completeness, quality and security of developed computer software. Several approaches under the tester point of view are used when software tests are designed [121].

After a vulnerability is discovered, there is often a delay before system administrators are notified, and additional time is needed to develop and deploy corrective patches. This window of exposure, which can last from hours to years, is frequently exploited by attackers. To address this, consistent methods for describing and sharing vulnerability information are crucial. The Common Vulnerabilities and Exposures (CVE) system developed by MITRE [114, 113] standardizes vulnerability naming and reporting but does not support their technical assessment. While signature-based detection methods are used in intrusion prevention and detection systems (IPS/IDS), the lack of robust standards for signature generation and representation limits their coverage [38]. The Open Vulnerability and Assessment Language (OVAL), as part of the SCAP protocol [24, 115], addresses this gap by offering a structured way to describe vulnerabilities using logical conditions observable in system configurations. Some research has extended this approach to address distributed vulnerabilities that span multiple systems [26].

Once vulnerabilities are described, they must be detected within systems and networks. Black-box approaches like network scanning and fingerprinting assess systems externally to infer information such as running services and software versions [9]. Grey-box methods go further by inspecting internal configurations, offering more accurate assessments. Using OVAL for vulnerability detection exemplifies a grey-box approach, as it encompasses system configuration modeling, vulnerability state analysis, and results reporting.

In addition to device-level analysis, assessing vulnerabilities across an entire network is essential, as attackers often exploit interdependencies. By modeling system states and potential attack paths, reasoning engines can infer vulnerabilities and support automated analysis. For example, MulVAL provides logic-based, multi-host vulnerability assessment capabilities [149].

Finally, the automation of remediation activities to bring the system back to a secure state is crucial in order to efficiently minimize the attack surface of the system over time. In this regard, techniques for evaluating the risks associated with changes, as the one proposed in [154], provide a support for vulnerability management, particularly for taking decisions about effective change implementations. Solutions such as [28] demonstrate the benefits of verification techniques in order to prevent new vulnerabilities when correc-

tive operations are executed. While some of these remediation approaches address distributed vulnerabilities, they do not consider the update process in itself, and even less its impact on congestion.

4.2.2 Service Placement

A key challenge in Edge Computing is the optimal placement of software units (i.e., virtual machines, containers, functions or services) of novel distributed applications. The research community has dedicated considerable effort to this topic in the last years and a vast number of theoretical results have been published addressing variations of this problem [87, 150, 158].

In this domain, the authors of [67] propose a two-time-scale framework for optimizing service placement and request scheduling in edge clouds for data-intensive applications. The approach achieves over 90% of optimal performance, effectively balancing stability and cost. Similarly, [105] introduces the RTSO algorithm to enhance service robustness in Edge-Cloud Computing under dynamic and unreliable conditions. The algorithm optimizes service placement by considering service popularity and resource constraints, achieving near-optimal performance in simulations.

Given the huge impact 5G had in the last years, the vast majority of works focus on Mobile Edge Computing as their reference scenario. Among them, [180] proposes a joint resource and service replica placement model in MEC to optimize resource utilization, energy consumption, and service availability. Using an improved NSGA-II algorithm with a two-phase chromosome representation, the study successfully minimizes resource waste and service unavailability compared to existing approaches. In the same direction, [165] introduces the LB-MCMF algorithm, a network-flow-based approach to optimize service placement and migration in MEC. It reduces latency and balances loads across edge nodes, outperforming traditional methods in dynamic user access scenarios.

Other works try instead to solve the placement problem in another groundbreaking scenario, that is, IoT. As an example, [167] proposes a Multi-objective Dynamic Service Placement (moDSP) algorithm to optimize response time and resource utilization in fog computing for real-time IoT applications. The algorithm, tested using iFogSim, significantly improves dead-

line compliance and resource use compared to other strategies. In [81], a Memetic Algorithm-based technique for optimizing application placement in fog and edge computing is introduced, focusing on reducing execution time and energy consumption for IoT applications. Finally, [156] proposes a fog computing framework to optimize IoT service placement, using a genetic algorithm to reduce communication delays and improve resource utilization. The approach outperforms centralized cloud solutions in these metrics.

On the other hand, there exist a limited number of works that aim to intersect service placement and security. Among these, the authors of [73] introduce SecFog, a methodology for securely deploying IoT applications across Cloud-Edge infrastructures by assessing security and trust. They use declarative programming to optimize deployment strategies, providing explainable security assessments. Later, the same authors introduce in [32] a declarative method for securely placing FaaS orchestrations in the Edge-Cloud Continuum, focusing on preventing data leaks. The approach, implemented in the SecFaaS2Fog tool, effectively balances security, execution times, and energy consumption. In [161], instead, a scheduling framework for edge clouds is introduced, balancing security and response times for time-sensitive applications. The proposed method reduces risk probability and processing delays, outperforming other approaches in simulations.

Table 4.1 provides a comparative overview of the most relevant approaches, highlighting their support for vulnerability evaluation, security-aware placement decisions, multi-objective optimization, adaptability to dynamic environments, and the consideration of service migrations in response to changing conditions. Although some existing works explore secure service placement, none explicitly consider vulnerabilities as a decisive factor in deployment decisions. Additionally, only a few approaches account for dynamic environments, and even fewer support service migration when deployment conditions change. Our proposed methodology addresses these gaps by integrating vulnerability assessment into the placement process and supporting adaptive service migration, thus enabling secure, flexible, and QoS-compliant deployments within the evolving Edge-Cloud Continuum.

Table 4.1: Qualitative Comparison of Existing Approaches.

Approach	F1	F2	F3	F4	F5
Sauve et al. [154]	✓	✗	✗	✗	✗
Farhadi et al. [67]	✗	✗	✓	∼	∼
Liu et al. [105]	✗	✗	✓	✓	✗
Woldeyes et al. [180]	✗	✗	✓	∼	✗
Tang et al. [165]	✗	✗	✓	✗	✓
Trabelsi et al. [167]	✗	✗	✓	✓	✗
Goudarzi et al. [81]	✗	✗	✓	✗	✗
Skarlat et al. [156]	✗	✗	✓	✗	✗
Forti et al. [73]	✗	✓	✗	✗	✗
Bocci et al. [32]	✗	✓	✓	✗	✗
Sun et al. [161]	✗	✓	✓	✗	✗
Our approach	✓	✓	✓	✓	✓

Legend: ✓= Yes, ✗= No, ∼= Partial/limited., F1 = Vuln. Assessment, F2 = Sec.-Aware Placement, F3 = Multi-Obj. Opt., F4 = Dynamic Environment, F5 = Migration Support

4.3 Methodology

Security is a complex and daunting matter to face due to the many perspectives from which it can be tackled. This becomes particularly true in Cloud-Edge, where security is an end-to-end goal covering from Cloud to On-premises. The Edge-Cloud Continuum represents a highly heterogeneous environment, characterized by a diverse array of facilities, each equipped with its own resources, security features, and flaws. Such flaws are predominantly introduced by the services deployed in the Continuum which inevitably carry vulnerabilities with them. Every deployed service has a vulnerability footprint given by the vulnerabilities affecting the service, e.g., software, network, and configuration vulnerabilities, all of which contribute to the security impact of the service. The combination of service footprints shapes the security of the facility on which they run.

The vulnerabilities introduced by a service can lower the security level of the facility hosting it. This has an impact on both the other services already

deployed on the same facility and on the subsequent deployment decisions. With respect to those other services, the security level reached by the facility can be low enough to make those services migrate to another facility with a more adequate security level. This can further induce other migrations, causing a ripple effect that can potentially pursue indefinitely. With regard to future deployment decisions, new services are less encouraged to be deployed on that facility, depending on their security requirements. If the security level of the facility gets excessively low, it may no longer be chosen for any new deployment until some of its services migrate or terminate and its security level is updated.

The two above cases show how the vulnerabilities of a service have an impact on the other services not only from a security point of view, but also from a deployment perspective. In particular, the services exposed the most to potential consequences of attacks are those directly interacting with the service carrying vulnerabilities, that is, those which have a dependency relation with the vulnerable service. Although containerization technology provides a method for isolating applications and their dependencies into self-contained units through a combination of kernel-level features and system-level configurations, there are known ways in which isolation can be circumvented, exploiting, for instance, kernel vulnerabilities, misconfigurations, shared resources and escape mechanisms [2]. As a consequence, even non-dependent services can be exposed to vulnerabilities.

Services have a significant influence on the tasks that utilize them, impacting dependent services directly. However, even unused services have a non-negligible security impact. For instance, a service under a CPU-intensive attack could monopolize resources, impeding other services from operating efficiently.

In this work, we introduce a methodology for optimal service deployment in the Edge-Cloud Continuum that takes into consideration security and migrations. It involves several steps, including identifying eligible solutions, evaluating cost profiles (that include both traditional QoS-related metrics and migration-oriented measures), and selecting the optimal placement. This methodology is illustrated in Figure 4.2, which outlines the interaction between key components of the deployment process. The process begins with the client submitting an annotated Application Workflow model. Si-

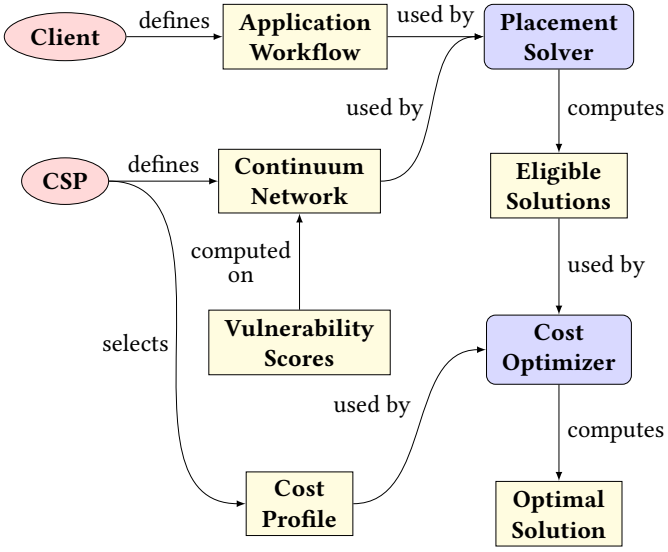

Figure 4.2: Cost optimized service deployment in the Continuum.

multaneously, the CSP defines its Continuum Network model, calculates facility vulnerability scores, and selects a cost profile. Using these inputs, the Placement Solver generates feasible solutions that meet vulnerability constraints, and the Cost Optimizer selects the optimal one based on the chosen cost metric. The Placement Solver computes a set of feasible solutions for the application deployment, which are then evaluated by the Cost Optimizer. In this work, we consider two types of cost: resource consumption, representing the amount of computational resources utilized by the deployed services, and migrations, quantifying the number of services that must be relocated as a consequence of the deployment decision. The optimizer uses the defined cost profile to identify the optimal solution, ensuring the service meets both functional and non-functional requirements. A profile specifies the degree to which a Cloud Service Provider (CSP) prioritizes migration costs, enabling dynamic adaptation of the deployment strategy at runtime. The purpose of employing different profiles is to mitigate the number of migrations by adjusting the weight assigned to migration costs within the deployment deci-

sion process.

A key aspect of our methodology lies in its ability to continuously monitor the vulnerability levels of deployment facilities over time. Since the security posture of a node can change due to evolving threat landscapes, updates to deployed services, or new vulnerabilities being discovered, we incorporate a dynamic assessment mechanism that regularly evaluates whether each node still satisfies the security requirements of the services it hosts. When a node's vulnerability level degrades below acceptable thresholds, the system proactively triggers service migrations toward more secure facilities. This adaptive response not only preserves the intended security guarantees but also prevents the accumulation of risk in specific areas of the Continuum, maintaining a balanced and resilient deployment landscape. To illustrate the proposed methodology, we outline a step-by-step example in the healthcare domain.

Example 4.1. *Modern healthcare systems rely on continuous patient monitoring to enable timely medical interventions. IoT devices play a key role in acquiring real-time physiological data, whose processing demands low-latency and secure infrastructures [3]. The Cloud-Edge Continuum offers a suitable hierarchical paradigm by distributing computational tasks across edge and cloud nodes. To give a real-world correspondence of the concepts in the proposed methodology, we consider a patient monitoring application as a guiding example, comprising a set of services designed for data collection, processing, storage, and visualization. Specifically, these services include: a data collection service (s_{dc}) responsible for acquiring, through AWS IoT Greengrass, raw physiological data from wearable sensors attached to patients; a data processing service (s_{dp}) tasked with analyzing the collected data, detecting anomalies, and generating alerts for healthcare providers using Apache Spark; a data storage service (s_{ds}) ensuring the secure preservation of processed data for future reference on a Oracle MySQL database; and finally, a data visualization service (s_{dv}), which provides healthcare professionals with real-time dashboards and notifications based on the processed data, leveraging Tableau Software. The deployment of this application is supported by a Cloud-Edge Continuum infrastructure, comprising an on-premise facility (f_{op}), two edge facilities (f_{e_1} and f_{e_2}), and two cloud facilities (f_{c_1} and f_{c_2}).*

4.4 System Model and Problem Formulation

This section delves into the methodology for assessing service vulnerabilities and for assigning a vulnerability score to each facility, providing a quantifiable measure of their security level. The proposed model for assessing vulnerabilities is built on the aggregation of individual service vulnerabilities and their potential interactions. We start by introducing the formal definitions of the elements of our methodology, first of all the annotated Continuum network. All the notation and symbols used in this chapter are reported in Table 4.2.

4.4.1 Annotated Continuum Network

In our model, grounded on and extending the methodology presented in Chapter 3, the facilities provided by a CSP are represented as a directed multigraph, where nodes model facilities and edges model communication links between facilities. To represent both the properties held and the resources provided in the network, each node and edge is annotated with one or more capabilities $c \in C$.

Definition 4.1 (Continuum network)**.** *A* Continuum network *is an annotated directed multi-graph* $N^C = \langle F, L, src_F, dst_F \rangle$ *consisting of:*

- *a finite set of* nodes *(*facilities*)* F*,*
- *a finite set of* edges *(*links*)* L*,*
- *a source mapping function* $src_F : L \mapsto F$ *assigning its source node to each edge,*
- *a target mapping function* $dst_F : L \mapsto F$ *assigning its target node to each edge.*

Example 4.2. *Considering Example 4.1, the Continuum network* N^C *comprises a set of 5 facilities, denoted as* $F = \{f_{op}, f_{e_1}, f_{e_2}, f_{c_1}, f_{c_2}\}$*, and a set of 14 links, represented as* $L = \{l_1, \dots, l_{14}\}$*. The Continuum network depicted in Figure 4.3*

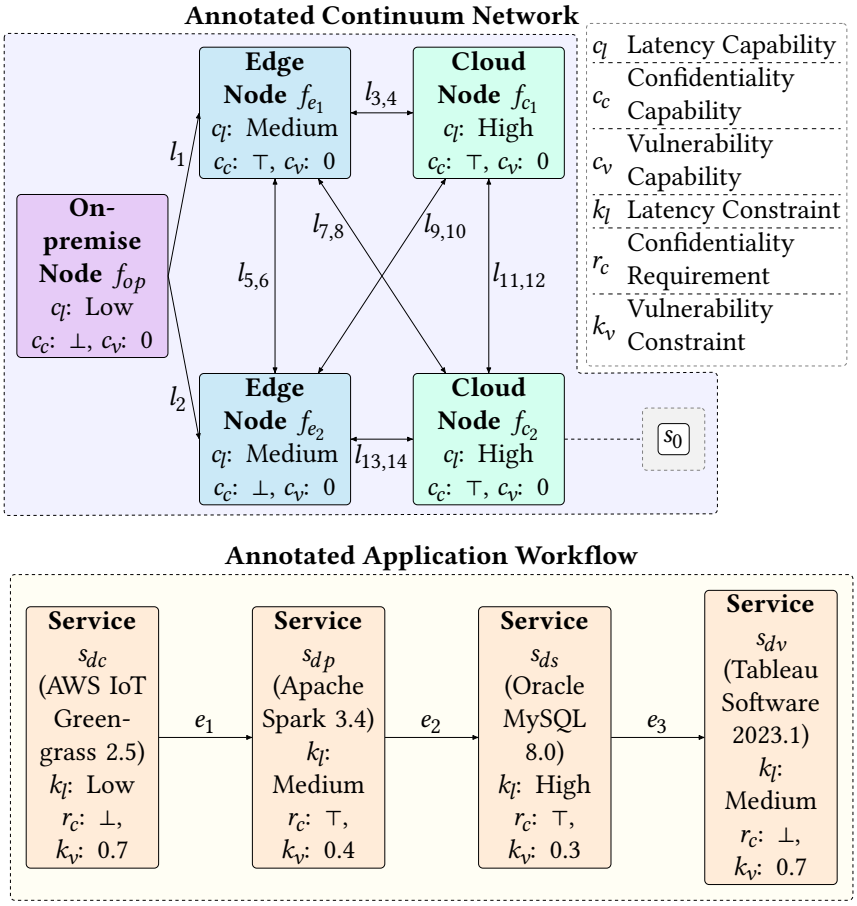

Figure 4.3: Patient monitoring application over the Continuum network.

graphically combines, for readability, bidirectional communication into bidirected edges, although edges are intended to have only one direction and be identified by one of the two labels associated with them in the figure. The mapping functions src_F and dst_F define the source and target of each link, respectively. For instance, $src_F(l_1) = f_{op}$ and $dst_F(l_1) = f_{e_1}$ specify that link l_1 originates from facility f_{op} and terminates at facility f_{e_1}.

Each element of the Continuum network has its own peculiarities, in terms of resources and properties, and can be configured differently from the others. Annotations reflect such differences by allowing the provider of the network to declare capabilities on what resources are provided and which properties are implemented.

Definition 4.2 (Capability Annotation)**.** *Let C be a finite set of capabilities and for each $c \in C$ let $Dom(c)$ be the set of possible values of the capability c. For each $c \in C$, the set $Dom(c)$ always contains the symbol $\perp$ (incompatible with other values), meaning that the capability c is not provided. A* capability annotation *is a function*

$$\gamma^C_{f \cup l} : C \to \cup_{c \in C} Dom(c) \tag{4.1}$$

defined for each facility $f \in F$ and each link $l \in L$ such that $\gamma^C_{f \cup l}(c) \in Dom(c)$ for all $c \in C$, i.e., $\gamma^C_{f \cup l}(c)$ returns the value of capability c provided by facility f or link l.

Example 4.3. *Both facilities and links are characterized by capability annotations. For the sake of this example, we consider two capabilities: the latency of a facility with respect to the end user (c_l) and the confidentiality at rest provided by a facility (c_c). The capability c_l is defined over the domain $Dom(c_l) = \{Low, Medium, High, \perp\}$, whereas the domain of c_c is given by $Dom(c_c) = \{\top, \perp\}$, where $\top$ indicates that confidentiality is ensured, while $\perp$ signifies its absence. In this context, the on-premise node offers low latency but no confidentiality, formally expressed as $\gamma^C_{f_{op}}(c_l) = Low$ and $\gamma^C_{f_{op}}(c_c) = \perp$. The edge nodes, f_{e_1} and f_{e_2}, provide medium latency, although only f_{e_1} ensures confidentiality. Regarding the cloud nodes, they exhibit both high latency and confidentiality at rest.*

In our problem we assume that each facility $f \in F$ has its own software configuration given by a list of installed software components with different software versions. A software component is any piece of software installed on a system, such as a program, package, or executable, that performs specific functions and typically has a version for tracking updates and compatibility.

Definition 4.3 (Facility Configuration)**.** *Let O be a finite set of software objects and for each $o \in O$ let $Dom(o)$ be the set of possible versions of the software object o. For each $o \in O$, the set $Dom(o)$ always contains the symbol $\perp$ (incomparable with the other software versions), meaning that the software object o is not installed. A* facility configuration *is a function*

$$\gamma_f^O : O \to \cup_{o \in O} Dom(o) \tag{4.2}$$

defined for each facility $f \in F$ such that $\gamma_f^O(o) \in Dom(o)$ for all $o \in O$, i.e., $\gamma_f^O(o)$ returns the version of the software object o that is installed on the facility f.

Example 4.4. *Furthermore, a facility can host a service, which runs a set of software objects with its own configuration. For instance, facility f_{c_2} hosts service s_0, which runs software object o_1 = Apache Kafka with a domain defined as $Dom(o_1) = \{3.0, 3.1, 3.2, \perp\}$. The notation $\gamma_{f_{c_2}}^O(o_1) = 3.0$ specifies that the installed version of o_1 on f_{c_2} is* 3.0*.*

4.4.2 Annotated Application Workflow

In the Continuum, applications are made of services organized in a workflow, which can be represented as a directed multi-graph. In this representation, nodes model services and edges model data flow. Symmetrically to facilities and links, the nodes and edges of a workflow are associated with annotations, which define requirements on the workflow.

Definition 4.4 (Application workflow)**.** *An* application workflow *is an annotated directed multi-graph $W^{R,K} = \langle S, E, src_S, dst_S \rangle$ consisting of:*

- *a finite set of* nodes *(*services*) S,*
- *a finite set of* edges *(*flows*) E,*
- *a source mapping function $src_S : E \mapsto S$ assigning its source node to each edge,*
- *a target mapping function $dst_S : E \mapsto S$ assigning its target node to each edge.*

The structure of a workflow imposes a logical dependency between services, which are represented as edges in the workflow graph. To preserve service dependencies, we ensure that any two connected services in the workflow are deployed on facilities that are linked in the same direction. This guarantees that required communications are supported in the resulting deployment.

Example 4.5. *Following Example 4.1, the application workflow $W^{R,K}$ consists of a set of services, denoted as $S = \{s_{dc}, s_{dp}, s_{ds}, s_{dv}\}$, and a set of flows, represented as $E = \{e_1, e_2, e_3\}$. The mapping functions src_S and dst_S define the source and destination nodes for each flow. For instance, for e_1, these mappings are given by $src_S(e_1) = s_{dc}$ and $dst_S(e_1) = s_{dp}$, indicating that flow e_1 originates from service s_{dp} and terminates at service s_{dp}.*

In a service-oriented application, each service has its own demands, in terms of resources and non-functional properties. Annotations reflect such demands by allowing users to define requests on the elements of the workflow. In this case, however, we distinguish between two kind of annotations: i) requirements $r \in R$, which refer to non-functional properties implemented by specific techniques (e.g., confidentiality enforced with AES encryption), and ii) constraints $k \in K$, which refer to measurable attributes that assess the resources provided by a facility. At a lower level, requirements refer to qualitative attributes and can assume only Boolean values (i.e., a property is implemented or not), while constraints refer to quantitative metrics and can assume also numerical values (e.g., a facility provides 32 GBs of memory).

Definition 4.5 (Requirement Annotation)**.** *Let R be a finite set of requirements and for each $r \in R$ let $Dom(r)$ be the set of possible values of the requirement r. For each $r \in R$, the set $Dom(r)$ always contains the symbol $\perp$ (incompatible with other values), meaning that the requirement r is not claimed. A* requirement annotation *is a function*

$$\gamma^R_{s \cup e} : R \rightarrow \cup_{r \in R} Dom(r) \tag{4.3}$$

defined for each service $s \in S$ and each flow $e \in E$ such that $\gamma^R_{s \cup e}(r) \in Dom(r)$ for all $r \in R$, i.e., $\gamma^R_{s \cup e}(r)$ returns the value of requirement r defined on service s or flow e.

Example 4.6. *Each service and flow is annotated with requirement specifications. Analogously to Example 4.1, we consider confidentiality at rest (r_c) as a requirement, defined over the domain $Dom(r_c) = \{\top, \bot\}$. Service s_{dc} is characterized by $\gamma^R_{s_{dc}}(r_c) = \bot$, as it does not require confidentiality, given that the collected data is raw and less sensitive. Instead, service s_{dp} requires confidentiality due to the sensitivity of the processed health data. The data storage service (s_{ds}) prioritizes confidentiality to protect stored sensitive information. Lastly, service s_{dv} does not impose confidentiality constraints.*

Definition 4.6 (Constraint Annotation)**.** *Let K be a finite set of constraints and for each $k \in K$ let $Dom(k)$ be the set of possible values of the constraint k. For each $k \in K$, the set $Dom(k)$ always contains the symbol $\bot$ (incompatible with other values), meaning that the constraint k is not claimed. A* constraint annotation *is a function*

$$\gamma^K_{s \cup e} : K \to \cup_{k \in K} Dom(k) \tag{4.4}$$

defined for each service $s \in S$ and each flow $e \in E$ such that $\gamma^K_{s \cup e}(k) \in Dom(k)$ for all $k \in K$, i.e., $\gamma^K_{s \cup e}(k)$ returns the value of constraint k defined on service s or flow e.

Example 4.7. *Each service and flow is also annotated with constraint specifications. We consider latency with respect to the end user (k_l) as a constraint, with a domain given by $Dom(k_l) = \{Low, Medium, High, \bot\}$. Service s_{dc} is characterized by $\gamma^K_{s_{dc}}(k_l) = Low$, as it necessitates minimal latency to ensure real-time data acquisition. Service s_{dp} requires low latency for prompt anomaly detection, while for the data storage service (s_{ds}) latency is less critical and can be high. Lastly, service s_{dv} requires medium latency to ensure timely and secure access to patient data.*

Similarly to facilities, a workflow has a configuration of the software objects included in its services given by the function

$$\gamma^O_s : O \to \cup_{o \in O} Dom(o) \tag{4.5}$$

defined for each service $s \in S$ such that $\gamma^O_s(o) \in Dom(o)$ for all $o \in O$, i.e., $\gamma^O_s(o)$ returns the version of the software object o that is included in the service s.

Example 4.8. *Each service specifies a software configuration that defines the versions of software objects on which it depends. Service s_{dc} runs o_2 = AWS IoT Greengrass with version* 2.5*, s_{dp} processed data with o_3 = Apache Spark version* 3.4*, s_{ds} depends on o_4 = Oracle MySQL version* 8.0*, and, finally, s_{dv} relies on version* 2023.1 *of software object o_5 = Tableau Software, which is formally expressed as $\gamma^O_{s_{dv}}(o_5) = 2023.1$.*

4.4.3 Vulnerability Model

For each service deployed on a facility, vulnerabilities from standard databases such as the Common Vulnerabilities and Exposures (CVE) and the National Vulnerability Database (NVD) are identified. In our approach, we consider any vulnerability that can be described using OVAL definitions, including version-based software flaws and misconfigurations. Given a facility $f \in F$ in a cloud network N^C, we say that the facility configuration γ^O_f is *vulnerable* if it contains a certain combination of software versions described, e.g., by OVAL vulnerability descriptions [115]. The OVAL specification is essentially a Boolean combination of atomic predicates over the statements about the state (e.g., version number, parameters) of the software components installed on a single facility.

Definition 4.7 (Software vulnerability)**.** *A vulnerability formula φ is a Boolean combination of atomic predicates of the form:*

$$\gamma^O_x(o) \bowtie state \tag{4.6}$$

where γ is the configuration function, $x \in F$, $o \in O$, $\bowtie \in \{\leq, =, \geq\}$ and $state \in Dom(o)$. We say that the vulnerability φ manifests *in a facility configuration γ^O_f with $f \in F$ in a network N^C and we note $\varphi \in \gamma^O_f$ if substituting $x.o \bowtie state$ with $\gamma^O_f(o) \bowtie state$ in all atomic predicates inside of φ results in evaluating the whole formula to true.*

Each vulnerability is associated with a severity score $\sigma(\varphi)$, typically on a scale from 0 to 10, as provided by the Common Vulnerability Scoring System (CVSS). Subsequently, the interaction between services on a facility are assessed to identify potential vulnerability chains. Services with network access or inter-service communication may compound vulnerabilities, leading

to a higher risk score. Then, the identified vulnerabilities and their severity for each facility are aggregated, considering both individual vulnerabilities and interaction-based vulnerabilities. This aggregation accounts for the diversity and severity of vulnerabilities, aiming to provide a holistic view of the facility security posture. Finally, the aggregated vulnerability score is normalized to a $[0, 1]$ scale, where 1 represents the highest level of vulnerability (least secure) and 0 represents the absence of vulnerabilities (most secure). The normalization process considers the maximum theoretical vulnerability score based on the number and severity of vulnerabilities assessed.

Definition 4.8 (Vulnerability score)**.** *Given a Continuum network* $N^C = \langle F, L, src_F, dst_F \rangle$ *with specific facility configurations* $\gamma_f^O : O \mapsto \bigvee_{o \in O} Dom(o)$ *for each* $f \in F$*, where* O *is a set of software objects, and a list of vulnerabilities* $\varphi_1, \ldots, \varphi_n$*, it is possible to compute the overall vulnerability score of a facility score:* $F \mapsto [0, 1]$ *as:*

$$score(f) = \frac{\sum_{1 \leq i \leq n \text{ s.t. } \varphi_i \in \gamma_f^O} \sigma(\varphi_i)}{\sum_{1 \leq i \leq n} \sigma(\varphi_i)} \tag{4.7}$$

The vulnerability score is a value characterizing each facility, manifesting to what degree a facility is affected by vulnerabilities or, in other words, reflecting its level of security. For this reason, it is included among the capabilities a facility can declare. As a consequence, clients can set constraints on the services they want to deploy grounded on the vulnerability score. A client may want that a service is deployed on a facility that guarantees at least a certain level of security. To achieve this, a vulnerability constraint in the $[0, 1]$ interval can be set on each service, meaning that the service must be deployed only on a facility that has a score equal or lower than the constraint. As an example, a user can claim a vulnerability constraint, defined as the lowest vulnerability score a facility can exhibit to host a service, as $k = 0.8$, which can be met only by those facilities whose vulnerability score is 0.8 or lower.

Example 4.9. *Let us consider a vulnerability* φ_1 *defined by a combination of two atomic predicates. Specifically, the vulnerability is characterized by the conditions* $\gamma_x^O(o_1) \leq 3.0 \wedge \gamma_x^O(o_4) \leq 8.0$*. The severity associated with the vul-*

nerability is given by $\sigma(\varphi_1) = 2$. *Since no facility hosts both objects included in* φ_1, *the vulnerability scores of all facilities is set to* 0.

Finally, the service placement problem can be defined. Given the list of vulnerabilities $\varphi_1, \ldots, \varphi_n$ from a standard database and considering i) $N^C = \langle F, L, src_F, dst_F \rangle$ to be an annotated network, ii) $W^{R,K} = \langle S, E, src_S, dst_S \rangle$ to be an annotated workflow, iii) R and K be the sets of all possible requirements and constraints, iv) C be the set of all possible capabilities, and v) $\gamma^O_{v^0} : S \mapsto \bigvee_{s \in S} Dom(s)$ for all $f \in F$ to be a set of initial facility configurations, the service placement problem can be expressed as computing a mapping function $M : S \to F$ such that for every service $s \in S$ in $W^{R,K}$ there exists a facility $f \in F$ in N^C satisfying that every requirement and constraint of s is met by a capability of f. In the case of a vulnerability constraint, it must occur that $score(f) \leq \gamma^K_s(k_v)$, where k_v denotes the vulnerability constraint and $\gamma^K_s(k_v)$ denotes the value of the vulnerability constraint on s. It is important to note that the vulnerability score reflects the security posture of a node, whereas the vulnerability constraint defines the maximum level of vulnerability that a service can tolerate.

Example 4.10. *The services described in Example 4.5 impose vulnerability constraints based on their sensitivity to potential security threats. Specifically, service* s_{dc} *exhibits a high tolerance to low security levels, as the data it processes remains in its raw form, with* $\gamma^K_{s_{dc}}(k_v) = 0.7$. *Conversely, services* s_{dp} *and* s_{ds} *require stringent security measures due to the sensitivity of the processed health data, with* $\gamma^K_{s_{dp}}(k_v) = 0.4$ *and* $\gamma^K_{s_{ds}}(k_v) = 0.3$, *respectively. Finally, service* s_{dv} *maintains a balanced security requirement, characterized by* $\gamma^K_{s_{dv}}(k_v) = 0.5$.

4.5 Vulnerability-driven Placement

Supporting user requirements provides an improvement in the overall security level of deployments since it enables automation in the implementation of the security features, which mitigates the risk of misconfigurations from users. At the same time, however, it places a hard limitation on the ability of the CSP to optimize deployment according to business policies. The CSP no longer has full control over the placement of services, which

in most cases can be deployed only on a subset of the available facilities. Nevertheless, optimization can still be performed on costs, since a combination of requirements generally produces multiple placement solutions. All possible solutions can then be compared based on business-relevant costs to choose the best placement from the CSP perspective. A CSP considers several costs when taking deployment decisions, including networking (use of resources such as backbone network, routers and switches), computational (CPU utilization and power consumption) and storage (amount of memory used) costs. Each cost is produced as an outgrowth of performing an activity, e.g., deploying, monitoring, and scaling services, which generally results in multiple costs to take into account. For instance, monitoring a service has an impact on performance through both CPU consumption and memory utilization. Among these activities, migration represents a preponderant source of cost in the life cycle of a service, both because it involves considerable resource usage (especially when data are involved in the migration) and because it causes a downturn in the service, which results in unavailability time for the user [85].

In our methodological setup, migration is triggered when a requirement is no longer met, which situation can occur whenever a change in the deployment environment takes place. This can happen frequently in a shared environment such as the Cloud-Edge Computing, especially if we consider the vulnerability score, which can change as frequently as new services are deployed on a facility. In this case, the more vulnerability constraints are broken at every deployment, the more migrations are triggered and the higher is the cost incurred by the CSP. This suggests that minimizing the number of vulnerability constraints violated when deploying a workflow has a direct impact on optimizing the cost of migrations. However, multiple costs can be optimized at the same time. In this work, we consider migration costs, central to dynamic deployment scenarios, and resource consumption costs, a widely adopted metric in the literature. However, the framework is extensible and can incorporate additional cost dimensions, such as communication-related costs, including inter-facility bandwidth usage and network delay penalties.

In the remainder of this chapter, let $W^{R,K} = \langle S, E, src_S, dst_S \rangle$ be an annotated workflow and $N^C = \langle F, L, src_F, dst_F \rangle$ be an annotated Continuum network. Let also R and K be the sets of all possible requirements and con-

straints, respectively, and C be the set of all possible capabilities.

4.5.1 Migration Costs

Migration costs are incurred when a service requirement is violated. A migration can be triggered by any change in the environment, both hardware and software. For instance, a hardware failure can break a constraint, while the expiration of a certificate can invalidate a requirement. If we take into account vulnerability constraints, a migration is triggered when the constraint value is lower than the facility capability. The *Instability Indicator* identifies when a vulnerability constraint is violated.

Definition 4.9 (Instability Indicator)**.** *Let $\gamma_s^K(k_v)$ be the vulnerability constraint value of service s and $score(f)$ be the vulnerability score of facility f. The Instability Indicator $I(s)$ is defined as:*

$$I(s) = \begin{cases} 1 & if\ \gamma_s^K(k_v) < score(M(s)), \\ 0 & otherwise \end{cases} \tag{4.8}$$

where $M(s)$ is the facility where service s is deployed.

The cost $\mathscr{C}_1$ of migrations triggered by invalidated vulnerability constraints can be computed as:

$$\mathscr{C}_1 = \sum_{s \in S} \frac{I(s)}{|S|} \tag{4.9}$$

$\mathscr{C}_1$ represents the cost of a deployment in terms of migrations caused by vulnerability constraint invalidation. It assumes that the cost of migrating each service is uniform. If two services have different migration costs (due to size, complexity, dependencies, etc.), this model can be extended by introducing a weight w_m for each service s, where w_m represents the relative cost of migrating service s.

Example 4.11. *According to the workflow requirements and constraints, only two feasible deployment solutions exist within the given environment. In both cases, service s_{dc} is assigned to facility f_{op}, s_{dp} to f_{e_1}, and s_{dv} to f_{e_2}. The primary difference lies in the assignment of service s_{ds}, which can be deployed on either*

f_{c_1} or f_{c_2}. If s_{ds} is assigned to f_{c_2}, this deployment necessitates running software object o_4 on that facility, thereby triggering the vulnerability condition of φ_1. Consequently, the vulnerability score of f_{c_2} increases from 0 to 1, violating the constraint $\gamma^K_{s_{ds}}(k_v) = 0.5$. As a result, service s_{ds} becomes unstable and requires migration after deployment. For this solution, the migration cost is given by $\mathscr{C}_1 = \frac{1}{4} = 0.25$. Alternatively, s_{ds} can be deployed on f_{c_1}, where no software objects are initially hosted. In this case, introducing object o_4 to f_{c_1} would not raise the facility's score, which remains within the acceptable range of the service constraint. Consequently, for this deployment scenario, the migration cost is $\mathscr{C}_1 = \frac{0}{4} = 0$.

As a business policy a CSP might want to preserve some of its facilities from vulnerabilities in order to keep them available for those services claiming a low vulnerability level. This can be expressed by a vulnerability upper bound, indicating what is the highest value the vulnerability capability of a facility can reach. If a deployment request makes the capability of a facility exceed its upper bound, the requested services deployed on that facility must be migrated. Such migration can involve all the newly deployed services or only the services causing the overflow (if they can be identified). To identify when a deployment makes a vulnerability level exceed the bound, we define the *Boundary Exceeding Indicator*.

Definition 4.10 (Boundary Exceeding Indicator)**.** *Let $score(f)$ be the vulnerability capability value of facility f and B^f_v be the vulnerability upper bound of facility f. The Boundary Exceeding Indicator $B(f)$ is defined as:*

$$B(f) = \begin{cases} 1 & \text{if } score(f) > B^f_v, \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

As a result the cost $\mathscr{C}_2$ of migrations triggered by vulnerability boundary exceeding can be computed as:

$$\mathscr{C}_2 = \sum_{f \in F} \frac{B(f)}{|F|} \tag{4.11}$$

As before, it assumes that the cost of migrations is uniform. However, being $B(f)$ computed on each facility and not on each service, it cannot be

weighted on the size or complexity of services. Nevertheless, a weight factor can be introduced in $\mathscr{C}_2$ build on the number of services in the current request deployed on the considered facility. Those services, or part of them, are the cause of the boundary exceeding and, thus, have to be migrated after deployment.

Example 4.12. *Let us consider a vulnerability upper bound for f_5, defined as $B_v^{f_{c_2}} = 0.3$. If s_{ds} is deployed on f_{c_2}, this upper bound would be exceeded, necessitating post-deployment migration of the service. Provided that no additional upper bounds are violated, the migration cost incurred due to exceeding the vulnerability threshold is $\mathscr{C}_2 = \frac{1}{5} = 0.2$.*

4.5.2 Resource Consumption Cost

Alongside migrations, a CSP might want to optimize the usage of its resources. Many factors can be considered to model resource consumption, including traditional parameters such as CPU, memory, and bandwidth, or more complex insights regarding energy consumption. As an example, we consider the number of active facilities (i.e., the facilities hosting at least one service) as a resource consumption parameter. However, other factors are equally possible. The set of active facilities U is given by:

$$U = \{M(s) \mid s \in S\} \tag{4.12}$$

Depending on its business policy, a CSP can minimize the number of active facilities (encouraging resource consolidation) or can maximize it (potentially for redundancy, fault tolerance, or load distribution purposes). To accommodate both options, the resource consumption cost $\mathscr{C}_3$ can be defined as:

$$\mathscr{C}_3 = d \cdot \left(1 - \frac{|U|}{|F|}\right) + (1 - d) \cdot \frac{|U|}{|F|} \tag{4.13}$$

where d is a distribution parameter set by the CSP which takes values in the range $[0, 1]$. Values closer to 0 foster configurations with fewer active facilities, while values closer to 1 favor configurations with more active facilities.

Example 4.13. *The resource consumption cost is also evaluated for the two deployment solutions presented in Example 4.11. In this context, we assume that the provider prioritizes maximizing the number of active facilities, and thus, the parameter d is set to* 0.7*. If service s_{ds} is deployed on facility f_{c_1}, the set of utilized facilities, U, includes all five facilities (since f_{c_2} is already allocated for s_0). Consequently, the resource consumption cost is computed as $\mathscr{C}_3 = 0.7 \cdot \left(1 - \frac{5}{5}\right) + (1 - 0.7) \cdot \frac{5}{5} = 0.7 \cdot 0 + 0.3 \cdot 1 = 0.3$. Conversely, if s_{ds} is deployed on f_{c_2}, the set U consists of only four facilities, leading to a resource consumption cost of $\mathscr{C}_3^{'} = 0.38$. Therefore, deploying s_{ds} on f_{c_1} is the preferable option, as it results in a lower resource consumption cost.*

4.6 Recursive Migrations Mitigation

Recursive migrations emerge as a pivotal challenge within our proposed methodology for service deployment in the Edge-Cloud Continuum. These migrations are triggered by the invalidation of deployment requirements, predominantly due to fluctuating vulnerability scores or environmental changes. Since migration are handled as deployments, this results in a domino effect, where one migration precipitates another, possibly undermining service stability and inflating operational costs. Addressing this cyclical dilemma is crucial for ensuring the robustness and efficiency of service deployments across the Edge-Cloud Continuum.

To mitigate the impact of recursive migrations, we introduce a dynamic weight function applied to the calculation of the total deployment cost. The total cost $\mathscr{C}_{tot}$ is defined as a combination of migration costs ($\mathscr{C}_1$ and $\mathscr{C}_2$) and resource consumption cost ($\mathscr{C}_3$), adjusted according to the deployment profile through weights:

$$\mathscr{C}_{tot} = w_1 \cdot f(\mathscr{C}_1 + \mathscr{C}_2) + w_2 \cdot \mathscr{C}_3 \tag{4.14}$$

Note that w_1 and w_2 assume values in the range $[0, 1]$, allowing to tip the balance towards migration or resource consumption costs respectively, and that $w_1 + w_2 = 1$.

The function $f(\mathscr{C}_1 + \mathscr{C}_2)$ allows to tackle the recursive migrations problem, by taking a different shape depending on four profiles:

- **Migration-Independent Profile**: This profile operates under the premise that migrations are an inherent aspect of Edge-Cloud Computing dynamic nature. It prioritizes performance and resource optimization without giving special consideration to migration costs. In this case,

$$f(\mathscr{C}_1 + \mathscr{C}_2) = 0 \tag{4.15}$$

- **Linear Migration Cost Profile**: Under this profile, migration costs are weighted linearly, creating a balance between the frequency of migrations and operational efficiency. The linear profile is given by

$$f(\mathscr{C}_1 + \mathscr{C}_2) = \mathscr{C}_1 + \mathscr{C}_2 \tag{4.16}$$

- **Exponential Migration Cost Profile**: Transitioning to an exponential weighting of migration costs, this profile aims to significantly penalize frequent migrations. The adjustment function becomes

$$f(\mathscr{C}_1 + \mathscr{C}_2) = (\mathscr{C}_1 + \mathscr{C}_2)^x \tag{4.17}$$

  with $x > 1$, enhancing the penalty on migrations exponentially.

- **No Migrations Profile**: The most conservative stance aims to eliminate migrations altogether, significantly impacting deployment choices to favor long-term stability over immediate performance gains. The function prioritizes configurations that minimize the potential for migrations:

$$f(\mathscr{C}_1 + \mathscr{C}_2) = \begin{cases} 0 & \text{if } \mathscr{C}_1 + \mathscr{C}_2 = 0, \\ \infty & \text{otherwise} \end{cases} \tag{4.18}$$

The CSP can adopt a progressive mitigation strategy, beginning with a migration-independent perspective, gradually adopting stricter profiles to reduce recursive migrations. This iterative refinement allows for flexible adaptation to varying operational contexts and requirements, minimizing migrations through progressive constraints on deployment decisions.

4.7 Experimental Evaluation

In this section, we experimentally evaluate our Cloud-Edge Continuum service deployment methodology, focusing primarily on optimizing deployment costs, which include consumed resources, vulnerability scores, and migration costs. Using a set of simulated requests and various migration profiles, we employ an exhaustive algorithm to calculate the total deployment cost (TDC) and the number of migrations (NoM). For each configuration, we assess: i) performance in terms of computation time, under different vulnerability constraints, ii) differences between the final deployments. Finally, we compare our approach with the state of the art to evaluate its effectiveness.

4.7.1 Experimental Setup

The experiments were run on a Windows® 11 machine equipped with an Intel® Core i7 @ 1.3 GHz CPU and 16 GB of RAM. The algorithm was implemented using Python 3.12.1. To evaluate TDC, NoM and computation time, we randomly generated 100 deployment requests, each encompassing from 2 to 10 services. The number of facilities in the simulated Continuum was 20. We simulated this scenario 5 times, considering the mean values for the metrics computation. To compute the difference between the final deployments of the four profiles, we considered sets of requests with incrementally growing size from 10 to 100 requests. Each request included from 2 to 15 services and, for each services, 2 to 5 facilities were eligible among the 20 available. This scenario was simulated 5 times and the mean values were considered.

The parameters defined in Section V are instantiated in the experiments as follows. Migration costs ($\mathscr{C}_1$ and $\mathscr{C}_2$) are computed by checking whether vulnerability constraints or thresholds are violated after deployment. The severity scores used for vulnerability computation are sampled from normalized CVSS values. For the resource cost ($\mathscr{C}_3$), we control the number of active facilities, using a distribution parameter $d = 0.5$, unless otherwise stated. These values are set uniformly across all simulations unless explicitly varied in specific scenarios.

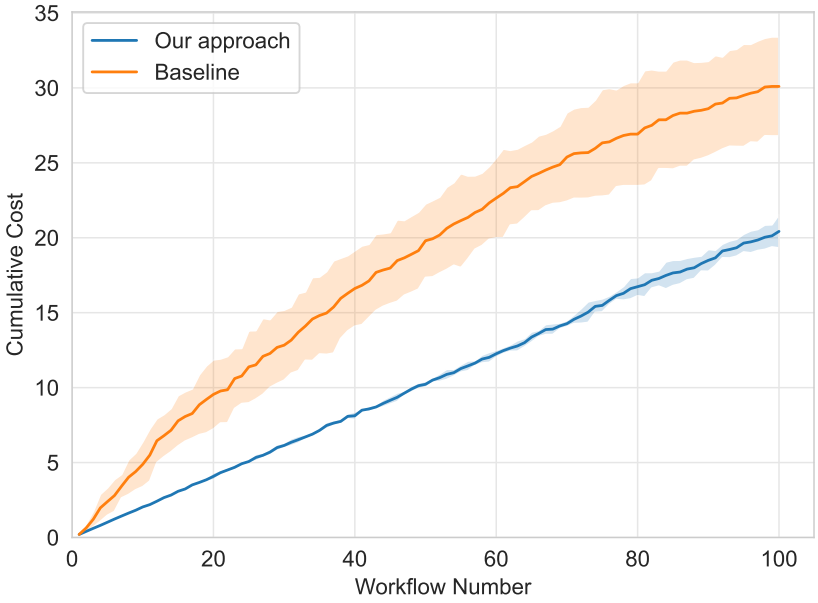

Figure 4.4: Mean cumulative cost over 100 requests with confidence bands.

4.7.2 Deployment Cost Assessment

This experimentation objective is to evaluate the effectiveness of the proposed methodology in reducing the cost of deployment. To do so, we collected in 5 executions the cumulative cost and the cumulative number of migrations over a batch of 100 simulated requests. To compute the cost we applied eq. (4.14) with both weights w_1 and w_2 equal to 0.5, and set $x = 2$ for the Exponential Migration Cost Profile. The applied profiles are incremented at every migration triggered. In other words, when a new request is handled, the Migration-Independent Profile is applied. If a migration is necessary, the Linear Migration Cost Profile is selected, and so on until the No Migrations Profile is applied. In this way, no more than three rounds of migrations can occur for each request. We compare the obtained results on the same set of requests with a baseline implemented by a stochastic approach that selects

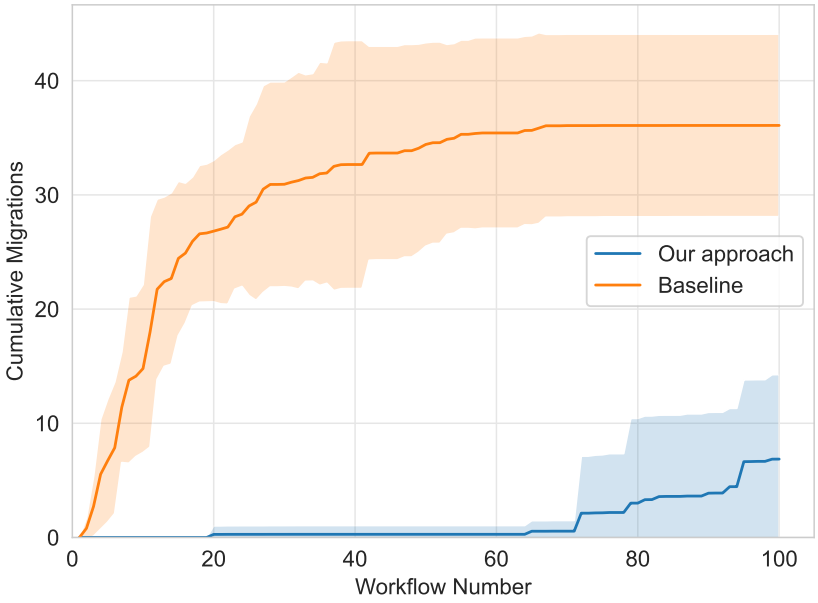

Figure 4.5: Mean cumulative number of migrations over 100 requests with confidence bands.

the deployment facilities randomly.

Figure 4.4 and Figure 4.5 show respectively the mean cumulative cost and the mean cumulative number of migrations for the two algorithms over 5 executions of 100 requests. The results show a substantial discrepancy on both cost and number of migrations, which starts to emerge from the first requests. On average, our approach menages to reduce TDC by 34.27%, compared to the baseline, and to reduce NoM by 80.61%. Referring to the number of migrations, we note that the gap between the two approaches is created in the first quarter of the experiment, while in the rest of it the trends tend to flatten. This is caused by the fact that, at the beginning of the experiment, the pool of possible solutions is larger and the baseline, selecting a random solution from the pool, is more likely to choose one causing migrations. In contrast, our service deployment methodology outperforms the baseline in

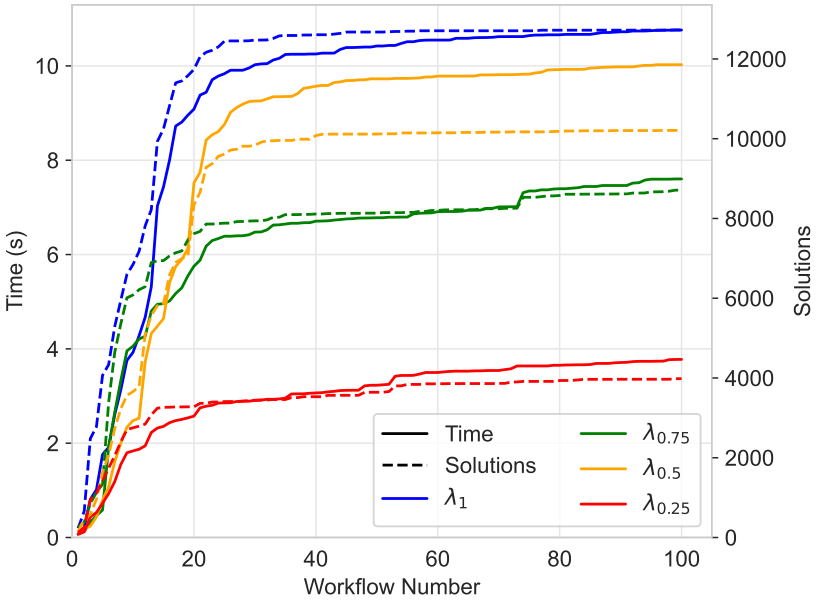

Figure 4.6: Deployment time and number of solutions over 100 requests, with different vulnerability constraint threshold λ. Time is in solid lines, number of solutions is in dashed lines.

the first part of the experiment, being able to identify the optimal deployment strategy in terms of NoM and, as consequence, also in terms of TDC.

4.7.3 Time Performance Evaluation

The focus of this experiment is to evaluate the performance of the proposed approach and the impact of different values of vulnerability constraints. To that effect, we generated four sets of services, each with a different vulnerability constraints threshold λ, i.e., $\lambda_{0.25} = 0.25$, $\lambda_{0.5} = 0.5$, $\lambda_{0.75} = 0.75$, $\lambda_1 = 1$. When the threshold is set to $\lambda_{0.25}$, for instance, the generated services are assigned with a vulnerability constraint $k_v^{s_i} \leq 0.25$. For each set of services, we assessed the computation time over 5 executions of 100 requests and col-

lected the time needed to accomplish each request. Additionally, we assessed the number of possible solutions for each request.

Figure 4.6 shows, for each λ, both the mean cumulative computation time (solid lines) and the mean cumulative number (dashed lines) of solutions over the requests. We note that, in general, a stricter λ leads to a decrease in the computation time. In particular, computation time with $\lambda_{0.5}$ is 9.03% lower than the one with no threshold (λ_1), while $\lambda_{0.75}$ takes 24.7% less than $\lambda_{0.5}$, and, finally, computation time with $\lambda_{0.25}$ is 49.84% lower then the time with $\lambda_{0.75}$. Such decrease is caused, as the overlapping patterns of time and number of solutions in Figure 4.6 underline, by an overall lower number of possible solutions for the services with a stricter vulnerability constraint, and the opposite for the looser ones. We note also that, for all values of λ, the majority of computation time is used for approximately the first 20 requests. After these, both patterns (time and solutions number) undergo a flattening. This demonstrates how, in this scenario, after a few deployments the number of possible solutions drastically decreases and, on the other hand, the number of missed deployments (i.e., the workflows that have no possible solution and are not deployed) raises.

4.7.4 Migration Profile Comparison

In this experiment, we aim at evaluating what variation in the deployment strategy is introduced by the four profiles. To do so, different sets of requests were generated with length varying from 10 to 100 requests (91 sets in total). To each set of requests, we executed our methodology four times, one for each profile. Then, we generated a deployment matrix $m = |S| \times |F|$, where $m_{i,j}$ is 1 if s_i was deployed on f_j, 0 otherwise. Subsequently, we computed a distance metric, in this case the Manhattan distance, between the Migration-Independent Profile deployment and the other three profiles. We replicated the simulation 5 times and take into account the mean values.

Figure 4.7 shows the resulting distances for the different length of the set of requests. Compared to the Migration-Independent Profile deployment (D1), the Linear Migration Cost Profile deployment (D2) and Exponential Migration Cost Profile deployment (D3) show similar deployment outputs, with the latter being on average slightly more different than the former. The

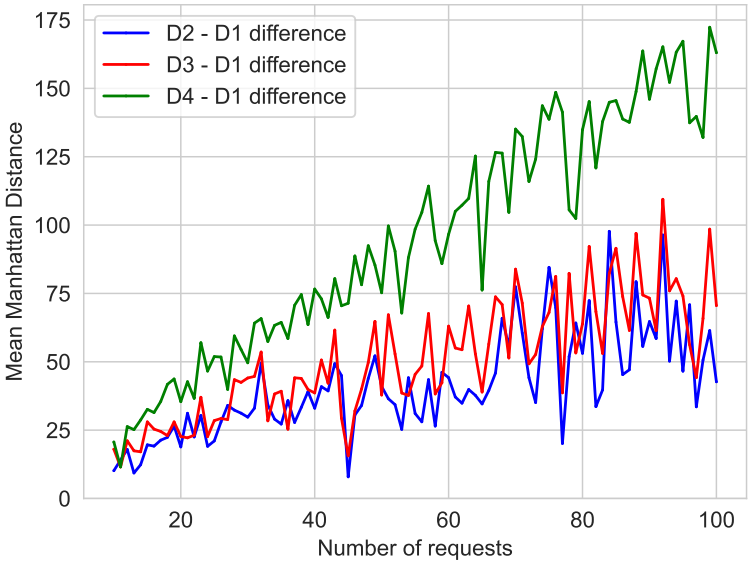

Figure 4.7: Mean Manhattan distance between Migration-Independent Profile deployment (D1) and Linear Migration Cost Profile deployment, Exponential Migration Cost Profile deployment (D3), and No Migrations Profile deployment (D4) for different sets of requests.

highest difference is exhibited by the No Migrations Profile deployment (D4), which implements a stricter behavior than the other profiles. It is worth noting that the difference gap between the D4 - D1 difference and the other two increases with the number of requests. This is intuitive, considering that the more services we introduce in the system, the more likely is it to migrate and, consequently, the more different will be the decisions taken by each profile.

4.7.5 State of the Art Comparison

To evaluate the effectiveness of the proposed approach in comparison to state-of-the-art service placement strategies, we conducted an experiment focused on NoM triggered by deployment decisions. In addition to our methodology, we considered the following benchmarks:

- *Stochastic* (ST) strategy, also employed in previous experiments, which randomly selects a deployment facility among the valid candidates;
- *MILP-based* approach from [67] (denoted as FA), which, given a batch of workflow requests, produces a placement by optimizing a target function. To align with the objectives of this study, we adopted $\mathscr{C}_3$ as the optimization criterion. Notably, this baseline does not account for service requirements or constraints;
- An extended version of the MILP-based (Ext-FA) approach that incorporates both service requirements and constraints.
- Latency-based Initial Placement (LIP) algorithm from [144]. It is based on a scoring method that sorts facilities using the networking delays and their price (i.e., the hourly price paid to use a server).

Consistent with the methodology adopted in previous experiments, we executed each approach over 100 simulated workflow requests, with $d = 0.5$ in $\mathscr{C}_3$, and recorded the resulting NoM. For the stochastic strategy, LIP and our proposed approach—being online strategies that process requests sequentially—the migration count is directly evaluated during execution. In contrast, FA and Ext-FA operate in batch mode, requiring a post-execution assessment to determine the number of triggered migrations.

Figure 4.8 presents the average NoM computed across five simulation runs, expressed as the percentage of migrated services over the total amount of services (381 on average). The objective of these strategies is to minimize both the number of migrations and missed deployments. The results indicate that the proposed approach significantly outperforms the baselines in terms of migration minimization, resulting in 2.6% of migrations and achieving a

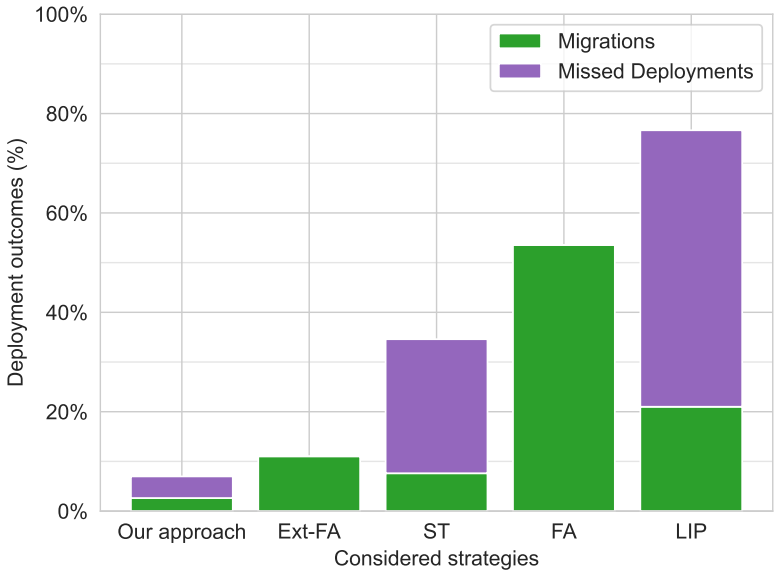

Figure 4.8: Average number of service migrations and missed deployments across different deployment strategies, with the objective of minimizing both of them.

reduction of 65% compared to the stochastic strategy (from 7.6%), 76% compared to Ext-FA (from 11%), 85.5% compared to LIP (from 20.8%), and 95% compared to FA approach (from 53.5%).

Nevertheless, it is important to note that FA and Ext-FA strategies successfully compute a deployment for the entire batch of workflows, whereas online methods may fail to deploy certain services when no valid assignment can be found. To highlight this trade-off, Figure 4.8 also reports the percentage of missed deployments, overlaid on the migration counts. Our approach results in 4.4% of missed deployments, in stark contrast to the 27% of failed deployments observed in the stochastic strategy and 66.4% in LIP. Missed deployments occur because the online approach handles requests individually, leading to resource exhaustion at needed facilities. Though unacceptable for critical applications, this can be mitigated by involving multiple CSPs or

scaling resources dynamically as failures arise.

4.8 Chapter Conclusions

This chapter introduces a comprehensive methodology for vulnerability-aware service deployment in the Edge-Cloud Continuum. By evaluating the vulnerability footprint of deployment targets and considering the impact of target platform vulnerabilities on the deployed service workflow, we outline a strategy that significantly enhances security posture and deployment efficiency. The proposed vulnerability assessment and scoring model, ranging from 0 (least secure) to 1 (most secure), provides an approach for evaluating node security in real-time, allowing for dynamic and informed deployment decisions. The implementation of this model into the deployment process enables a more secure, efficient, and adaptable Edge-Cloud Continuum. It addresses a critical gap in current practices by providing a mechanism to quantify and mitigate the risks associated with service deployment, thereby improving the overall trustworthiness and reliability of distributed computing environments. As future work, we plan to further integrate the vulnerability assessment and scoring mechanism with existing service deployment tools and workflows, to automate and streamline the deployment process, and to investigate advanced optimization approaches in order to further improve the deployment decision process.

While our methodology provides a formal and practical framework for vulnerability-aware service deployment in the Cloud-Edge Continuum, several limitations remain. First, the current model assumes a centralized decision-making process, which may not scale efficiently in large, distributed environments. Second, our approach relies on static vulnerability assessments based on predefined OVAL definitions, potentially limiting adaptability to emerging threats. Third, we have not yet integrated our methodology with existing orchestration frameworks, which could facilitate broader adoption.

In the following chapter (Chapter 5), we leverage the provided framework to build a workflow definition language that allows to express service requirements and adjust the final deployment on the specific property implementation of the target facility.

Table 4.2: Table of Notations.

Notation	Section	Definition
N	4.4.1	Continuum network
F	4.4.1	Set of Continuum facilities
L	4.4.1	Set of links between facilities
$src_F(l)$	4.4.1	Source node of link l
$dst_F(l)$	4.4.1	Destination node of link l
C	4.4.1	Set of facility capabilities
$Dom(x)$	4.4.1	Domain of variable x
$\gamma^C_{f \cup l}(c)$	4.4.1	Value of constraint c annotated on facility f or link l
O	4.4.1	Set of software objects
$\gamma^O_f(o)$	4.4.1	Software configuration on facility f
W	4.4.2	Application workflow
S	4.4.2	Set of services
E	4.4.2	Set of flows
$src_S(e)$	4.4.1	Source service of flow e
$dst_S(e)$	4.4.1	Destination service of flow e
R	4.4.2	Set of service requirements
K	4.4.2	Set of service constraints
$\gamma^R_{s \cup e}(r)$	4.4.2	Value of requirement r annotated on service s or flow e
$\gamma^K_{s \cup e}(k)$	4.4.2	Value of constraint k annotated on service s or flow e
$\gamma^O_s(o)$	4.4.2	Software configuration of service s
φ	4.4.3	Software vulnerability
$\sigma(\varphi)$	4.4.3	Severity of vulnerability φ
$score(f)$	4.4.3	Vulnerability score of facility f
$I(s)$	4.5.1	Instability indicator of service s
$B(f)$	4.5.1	Boundary exceeding indicator of facility f
$\mathscr{C}_1$	4.5.1	Cost of vulnerability constraint invalidation
$\mathscr{C}_2$	4.5.1	Cost of boundary exceeding
U	4.5.2	Set of active facilities
$\mathscr{C}_3$	4.5.2	Cost of resource consumption
$\mathscr{C}_{tot}$	4.6	Total deployment cost

5

Polymorphic Workflow Description Language

This chapter presents a polymorphic workflow description language for the Cloud-Edge Continuum. The language extends traditional workflow models with annotations and choreography-based constructs, supporting adaptive, secure, and efficient deployment of service compositions. It enables developers to define workflows in a way that is both expressive and execution-ready, bridging the gap between abstract specifications and practical deployment. The work presented in this chapter moves towards the materialization of the abstract framework introduced in Chapters 3 and 4 by proposing an implementation of the deployment process based on containers. The description language inherits from our graph-based representation of workflows, building a concrete example application. The content of this chapter is based on the following publication: *R. Bondaruc, A. Ovena, M. Perfumo, M. Gala, E. Damiani, and M. Anisetti, "Secure and Polymorphic Composition of Service Workflows in the Cloud Continuum", in 17th International Conference on Management of Digital Ecosystems, 2025.*

5.1 Introduction

The specification of workflows is central to the execution of complex, distributed applications. Traditional workflow description languages (WDLs), such as BPEL, were designed for centralized, cloud-based orchestration and struggle to capture the decentralized, heterogeneous, and dynamic nature of

the Cloud-Edge Continuum. They often lack constructs for runtime adaptability, secure annotations, and efficient representation, limiting their ability to support property-driven deployment of modern service compositions.

The motivation for this chapter stems from the need for a workflow language that goes beyond static orchestration and embraces the polymorphic, distributed reality of continuum computing. Existing WDLs either provide limited expressiveness or are too rigid to accommodate context-driven deployment decisions. In particular, they fail to link abstract service specifications with the constraints and capabilities of continuum facilities, forcing developers to bridge the gap manually.

The goal of this chapter is to design a polymorphic WDL tailored for the Cloud-Edge Continuum. The proposed language introduces annotations for non-functional requirements, supports choreography-based interaction models, and integrates with a polymorphic deployment engine capable of interpreting workflows and generating deployment-ready instances. This allows developers to specify workflows at a high level while ensuring that they can be executed efficiently and securely across heterogeneous infrastructures.

The contributions of this chapter include: i) a polymorphic methodology for extending workflows with QoS-aware annotations, ii) a choreography-based WDL that captures decentralized interactions while supporting secure and adaptive execution, and iii) an experimental validation showing improvements in representation efficiency and runtime performance compared to existing WDLs.

5.1.1 Chapter Outline

The remainder of the chapter is organized as follows. Section 5.2 presents our polymorphic service composition methodology. Section 5.3 illustrates the architecture implementing our methodology. Section 5.4 introduces cWDL, our polymorphic modeling language for choreographic workflows. Section 5.5 discusses the experimental evaluation of the proposed methodology. Finally, Section 5.6 outlines the conclusions.

5.2 Polymorphic Methodology

In this chapter, we consider a scenario in which: i) users aim to design and deploy data processing workflows composed of various tasks, and ii) a Cloud Service Provider (CSP) offers its Continuum (CSP Continuum) for workflow deployment, supporting workflow design through a platform such as the one proposed by MUSA [12]. A workflow is defined as a composition of services organized in a graph structure, where each node represents a service. By definition, a workflow encompasses the concepts of sequentiality (i.e., a service must be executed before others), parallelization (i.e., multiple services can be executed concurrently), and inter-service communication, represented by the edges of the graph.

To compose their workflows, users generate a workflow deployment request by selecting a set of services from a registry, that is, an indexed catalog containing available services, their images, and functional descriptions. Services are selected based on the operations they implement, the manner of the implementation, and output formats. After selecting a service, users may annotate it with non-functional requirements that specify the resources and properties expected from the facility that will host the service. Non-functional requirements include QoS constraints (e.g., latency, execution deadlines, resource demands), as well as privacy and security-related constraints, such as confidentiality, integrity, and authentication.

Once the workflow request is completed and annotated, it is submitted to the PDE, which computes the optimal placement for each service and orchestrates the deployment across the CSP Continuum. This Continuum consists of a set of computational nodes arranged in a graph, with each node representing a facility. Due to the potential limitation in reachability across the infrastructure, the edges of the graph denote the available communication links between pairs of facilities. Each facility is annotated with its capabilities, i.e., the resources it hosts and the security properties it guarantees.

Non-functional properties can be provided by a facility in one of two ways: inheritance or overriding. Inheritance applies when deploying a service on a facility inherently satisfies the required property. In other words, no additional mechanism is required to ensure the requested property other than selecting that specific node for the deployment. For instance, confiden-

tiality at rest is inherited when a facility employs an encrypted file system, as any service using that system automatically benefits from the property. In contrast, overriding is required when the desired property cannot be guaranteed solely by the deployment node and necessitates additional operations. This typically involves modifying the workflow with the injection of one or more support services, following defined patterns (discussed in Section 5.4.3). A support service is thus a service originally not part of the workflow, but is added by the PDE to enforce a property. An example of overriding is the enforcement of integrity, achieved by injecting a service that computes a cryptographic signature of the data before it is processed by the target service.

The ability to override the workflow endows our approach with polymorphism, enabling the workflow structure to adapt to varying deployment contexts. Specifically, our methodology supports: i) describing the workflow (Θ) by selecting tasks using our cWDL language; ii) annotating tasks, inter-task channels, and the entire workflow with non-functional requirements to yield an enriched description (Θ'); and iii) generating a deployable workflow instance ($\overline{\Theta}$) along with its corresponding deployment recipe ($\overline{\Theta}_r$) for execution on the given CSP Continuum.

Example 5.1 (Reference Scenario)**.** *Consider a pharmaceutical company aiming to perform molecular development via molecular-specific simulation workflows. Assume the company has an agreement with a given CSP implementing our PDE. According to these agreements, the CSP Continuum may include nodes located on the pharmaceutical premises (e.g., for data collection) and is utilized for deploying the company's workflows. Given the sensitive nature of pharmaceutical research, the following non-functional properties must be guaranteed: i) only authorized users are permitted to execute specific analytics on designated molecules; and ii) fine-grained control is enforced over the integrity, security and intellectual property of the developed molecules.*

5.3 Architecture

Figure 5.1 illustrates the architecture implementing the methodology introduced in Section 5.2. It is composed of: i) a registry that stores the set of

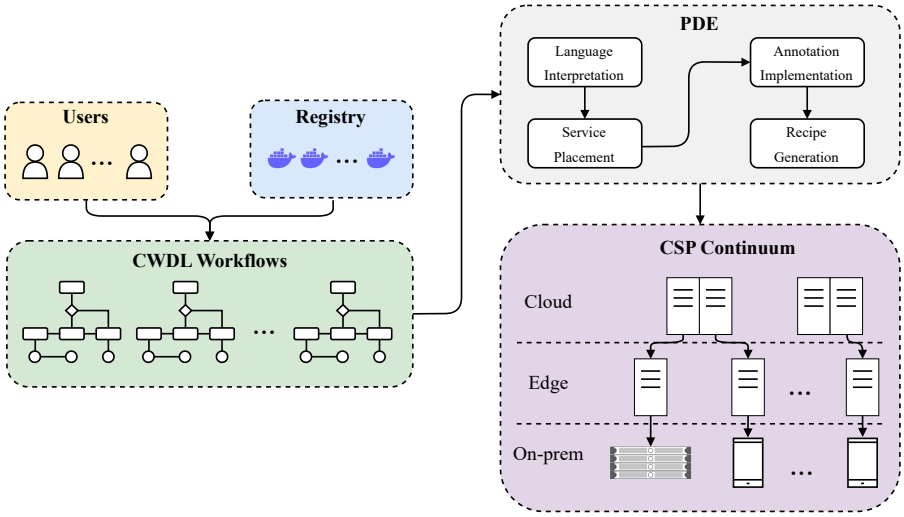

Figure 5.1: Deployment architecture including a Polymorphic Deployment Engine (PDE) interpreting annotated workflows and deploying them in the Cloud-Edge Continuum.

available tasks for workflow composition; ii) a high-level description of the workflow Θ and its annotated version Θ' to be deployed; and iii) the PDE engine, responsible for generating, deploying, and executing the workflow on the CSP Continuum using the underlying CSP deployment technology.

The PDE engine is the core component of the architecture and operates through four main phases. First, in the language interpretation phase, it parses the annotated workflow Θ' expressed in cWDL to verify its syntactic and semantic correctness (e.g., ensuring the workflow is connected and well-formed). Second, it performs service placement, computing an execution plan that satisfies both functional and non-functional requirements. Third, during the annotation implementation phase, it applies polymorphic transformations to the workflow to enforce non-functional requirements that cannot be fulfilled by existing tasks. For example, if authentication is required but not natively supported by any task, the PDE engine injects an appropriate authorization service, thereby producing the workflow instance $\overline{\Theta}$. Finally, in the recipe generation phase, it creates a deployment artifact $\overline{\Theta}_r$ tailored to

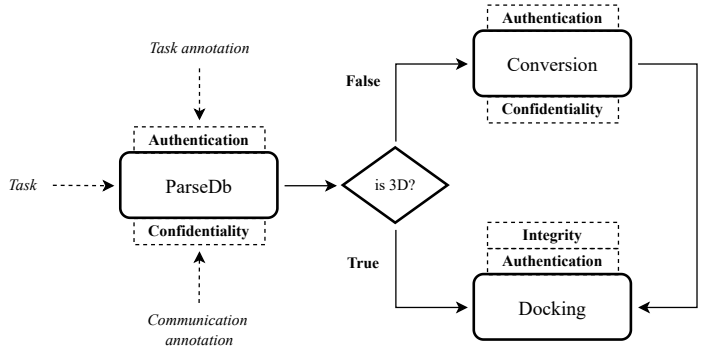

Figure 5.2: Θ′ workflow for our scenario in Example 5.2. The workflow is represented in a graphical form, with annotations depicted as dotted rectangle attached to the relevant tasks.

the target environment (e.g., Docker Compose, Kubernetes, AWS Step Functions, HPC clusters), enabling automated deployment and execution.

Example 5.2 (Θ and Θ′)**.** *Consider the scenario introduced in Example 5.1, where a pharmaceutical company designs a molecular docking workflow. The process begins with the selection of tasks to construct the workflow* Θ*. First, the ParseDb task retrieves data from the company's databases. Next, the Conversion task transforms two-dimensional molecular representations into three-dimensional structures. Finally, the Docking task executes the molecular docking process. The client then annotates the entire workflow with an authentication requirement, adds channel confidentiality annotations to ParseDb and Conversion, and applies an integrity annotation to Docking, thus producing the annotated workflow* Θ′*, illustrated in Figure 5.2.*

5.4 Choreography-based WDL Language

In this section, we introduce cWDL, our proposed language for defining choreographic workflows of services within the polymorphic methodology. cWDL builds upon WDL by adopting a choreographic execution model and incorporating support for security annotations. Unlike orchestration, chore-

ography eliminates the need for a centralized entity managing the data flow; instead, the control is distributed among the services themselves. The grammar of cWDL is derived from the OpenWDL standard, from which it inherits most structures and keywords [172]. The main differences lie in the definition of workflows and the mechanism used to connect tasks. In cWDL, tasks are modeled as services that not only implement the functional logic but also manage communication with subsequent tasks. Thus, workflows are expressed as chains of tasks in which each task explicitly defines the next step upon completion.

A choreographic workflow consists of two sections: definitions and tasks. The definitions section serves as the workflow header and includes metadata necessary for workflow execution, such as version, options, dependencies, data structures, and the starting task. Data structures are specified in a C-like syntax, enabling precise definition of the data exchanged between tasks. The starting task is declared in this section using the *start* construct. The tasks section describes how each service is built and executed, and supports security annotations.

5.4.1 Tasks

A task is composed of several sections: input, command, runtime, and next. The input section defines the task inputs and constants, specifying their data types and names, as well as any constant values. The command section lists the shell commands to be executed once the inputs are provided, and supports an escape syntax for referencing variables. The runtime section contains key-value pairs specifying the execution environment, including container runtime and image details. The next section determines the subsequent task to invoke after execution, using the call construct to map outputs to the inputs of the next task. Variables, constants, and conditional logic can also be used via the *if* construct to define conditional workflows. If a task produces the final output of the workflow, it includes a result section specifying the result structure, which is returned only if no subsequent task is invoked.

Example 5.3 (Task definition)**.** *In the* Θ *workflow from Example 5.2 expressed in cWDL, the ParseDb task takes a File input named Db, and its output is aligned*

with the requirements of the subsequent task, to which it is forwarded as input. Additionally, the molecules struct (defined in the workflow header) is instantiated as data. This variable is passed to the next task (Conversion or Docking) via the call construct, alongside Db. The logic of ParseDb is defined in the command section, where data is extracted from Db using the syntax ~ {Db}.

5.4.2 Annotations

cWDL supports two types of annotations on a workflow Θ: transformation annotations and constraint annotations. Transformation annotations, declared using the *with* keyword, modify the workflow structure (e.g., by adding tasks to enforce polymorphic behaviors). Constraint annotations, specified via the *runtime* or *comms* keywords, define execution parameters (e.g., performance or resource requirements) or communication properties (e.g., secure channels between tasks).

Example 5.4 (Task annotation)**.** *Authentication is enforced globally using the with auth annotation on the start call. Encryption between tasks is configured through the comms section, where confidentiality is set and HTTPS is specified as the secure protocol. Furthermore, the ParseDb task includes constraint annotations in its runtime section, detailing the container version, engine, node type, and RAM allocation.*

5.4.3 Workflow Instance Generation

The annotations specified within the workflow are processed by the PDE engine, which handles both constraint annotations and transformation annotations. Constraint annotations are evaluated in a subsequent stage, immediately prior to recipe generation, to guide deployment decisions. Transformation annotations, instead, are translated by the PDE into predefined patterns, classified as follows: i) global patterns, which affect all tasks in the workflow (e.g., the auth pattern in Example 5.4); ii) intra-task patterns, which modify only the annotated tasks; and iii) inter-task patterns, which affect interactions between the annotated task and one or more downstream tasks (e.g., the HTTPS pattern in Example 5.4).

As an example of an intra-task pattern, an availability pattern introduces task replication within the workflow. This approach is fully transparent to other tasks, thereby preserving a black-box model in which intra-task modifications remain encapsulated and isolated from the broader composition.

Example 5.5 ($\overline{\Theta}$)**.** *Consider the annotated workflow Θ' described in Example 5.2. The PDE engine applies the annotations to generate the instantiated workflow $\overline{\Theta}$. In this case, the global authentication annotation in Θ' is implemented by adding an authentication task at the beginning of the workflow to provide an authentication mechanism (e.g., certificate-based or OAuth-based). Additionally, each task is augmented with authentication features by injecting authentication logic into their container images, enabling them to authenticate against the authentication service. The PDE engine also enforces channel encryption for the ParseDb task and appends a signature-verification task after Docking to ensure data integrity.*

5.5 Experimental Evaluation

We evaluated the proposed language along two primary dimensions: workflow representation efficiency (measured by the length of the workflow description) and runtime performance (measured by total execution time). This evaluation highlights the benefits of our choreography-based PDE in comparison to existing solutions built on WDL and the Common Workflow Language (CWL) [51]. For this purpose, we selected three representative workflows with distinct complexity and operational characteristics: i) a molecular docking workflow following the reference scenario in Example 5.1, ii) a code pipeline simulating a typical DevOps automation process, iii) an MLP data pipeline for model training and evaluation. The code pipeline involves three services to download data from a source, process it, and upload it to a destination, respectively. The MLP data pipeline features five services to download the training dataset, perform preprocessing, train the model, evaluate it, and upload the results. These workflows encompass realistic use cases with diverse computational structures, data dependencies, and security requirements, allowing us to assess the flexibility and generality of our approach. Before evaluating runtime performance, we illustrate the process of recipe

generation in a Docker-based deployment environment. Additionally, we report on preliminary experiments evaluating the PDE engine's ability to generate deployment artifacts for HPC environments, thereby demonstrating the platform-agnostic nature of our methodology. In this context, we consider MLP-based data-intensive workflows derived from the UAERep project [25, 4] as a representative benchmark for HPC execution.

5.5.1 Experimental settings

All experiments were conducted on a virtual machine provisioned with 4 vCores and 6 GB of RAM, hosted on a system equipped with an Intel® Core i5-10400 CPU running at 2.90 GHz and 3200 MHz RAM. Each workflow was executed 100 times to ensure statistical robustness. Execution times were measured from workflow submission to completion, encompassing all orchestration and task execution phases. To mitigate the influence of anomalies, outlier values exceeding two standard deviations from the mean were discarded prior to analysis, ensuring that the reported results reflect stable and representative performance trends.

5.5.2 Representation efficiency

Figure 5.3 reports the number of lines of code required to describe the three workflows using WDL, CWL, and cWDL. Overall, code length across the three workflows ranges from approximately 75 to 150 lines. It can be observed that CWL consistently results in the most verbose implementations, reflecting its higher syntactic overhead and lower level of abstraction. In contrast, WDL and cWDL exhibit comparable conciseness, which is expected given that cWDL extends WDL by adapting it to a choreographic and pattern-based paradigm.

In the first experiment, with *Molecular Docking* workflow, CWL required approximately 150 lines of code, making it the most verbose among the three. WDL reduced this to around 120 lines, offering a more compact specification. Notably, cWDL produced the shortest implementation, with approximately 95 lines. This reduction is primarily attributed to the use of predefined patterns (e.g., the authentication pattern), which eliminate the need for developers to manually implement recurring non-functional features such as au-

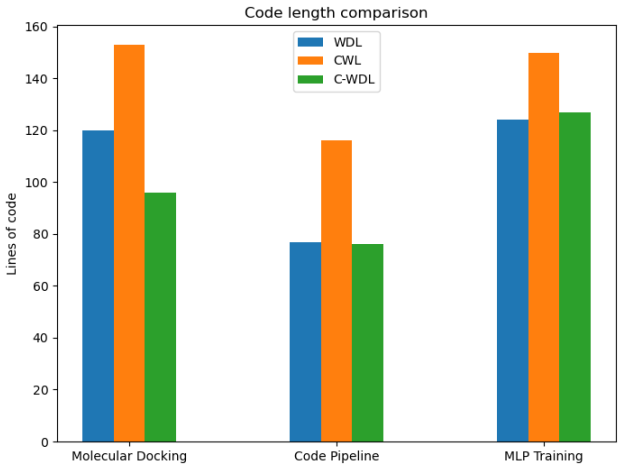

Figure 5.3: Code length comparison.

thentication. By abstracting these concerns, cWDL simplifies workflow design, standardizes the handling of cross-cutting properties, and reduces the potential for errors or inconsistencies.

In the second experiment, *Code Pipeline*, which involves a simpler workflow, WDL and cWDL resulted in nearly identical code lengths. This outcome reflects their shared syntactic foundation: cWDL preserves the readability and simplicity of WDL while offering additional expressiveness through optional patterns. Conversely, CWL again required substantially more lines of code, reinforcing its verbosity even for straightforward workflows.

Finally, in the third experiment, *MLP Training*, we evaluated the languages in a more complex, data-intensive context. Here, cWDL was implemented without the use of patterns, resulting in a code length similar to that of WDL. This demonstrates that cWDL is capable of maintaining the core advantages of WDL even in pattern-free scenarios. More importantly, it underscores cWDL's flexibility: developers can selectively apply patterns when beneficial, while still leveraging a syntax that remains concise and consistent for workflows that do not require them.

Table 5.1: Comparison of execution times across different languages.

Experiment	Metric	cWDL	WDL	CWL
Code Pipeline	Mean (ms)	9863.57	38668.97	7588.72
	Min (ms)	8240.00	33235.00	6680.00
	Max (ms)	13416.00	46351.00	10505.00
	CV (%)	11.57	7.37	13.10
Molecular Docking	Mean (ms)	8460.66	6764.70	7203.21
	Min (ms)	3753.00	4024.00	6002.00
	Max (ms)	22290.00	11235.00	13173.00
	CV (%)	33.58	23.21	21.96
MLP Data	Mean (ms)	8260.62	43626.79	10630.31
	Min (ms)	6476.00	37397.00	8820.00
	Max (ms)	13586.00	51033.00	22512.00
	CV (%)	18.43	7.03	21.02

CV (%) represents the Coefficient of Variation, calculated as the standard deviation divided by the mean, expressed as a percentage.

5.5.3 Runtime performance

Execution time is a critical metric for evaluating workflow systems, particularly in the Cloud-Edge Continuum, where latency, resource constraints, and overall completion time significantly impact performance. While our previous analysis examined code length as a measure of language readability and maintainability, runtime performance directly determines the feasibility of deploying workflow solutions in production environments. The choreographed execution model of PDE, in contrast to the orchestration models of WDL and CWL, introduces a fundamentally different paradigm whose performance warrants careful evaluation.

This experiment aims at evaluating the impact of the choreographed model on runtime performance compared to orchestration-based alternatives. An additional distinction is that PDE generates a Docker Compose file that can be executed manually, offering operators the flexibility to inspect

and modify it prior to execution. For comparability in our experiments, we used a Bash script to automate the execution of the PDE-generated composition, thereby aligning it with the execution methods used for WDL and CWL. The three benchmark workflows employed in our study reflect common real-world computational patterns. The *MLP Data* and *Code Pipeline* workflows share elements such as file transfers via *curl*, whereas the *Molecular Docking* workflow emphasizes compute-intensive tasks without network transfer overhead.

Table 5.1 presents the mean, maximum, and minimum execution time statistics for the three workflows across all tested languages. Several notable patterns emerge from this data. Overall, it can be seen that cWDL and CWL perform similarly across all experiments, while WDL shows a significant slowdown compared to the others. This anomaly, only present in Code Pipeline and MLP Data, could be caused by the use of web requests, as the task that took the longest to complete was *DownloadFile*. Given that identical commands were used across all languages, this bottleneck highlights differences in execution handling between the paradigms. With reference to the code pipeline, cWDL and CWL significantly outperform WDL, due to the limitations discussed above. In particular, cWDL and CWL yield comparable results, with CWL demonstrating marginally superior performance. In the Molecular Docking experiment, which operates without web requests, WDL achieves the best performance with the fastest execution times. CWL and cWDL maintain equivalent performance characteristics throughout these tests. In the case of MLP Data experiments, the results align with the findings on code pipeline, with cWDL being the fastest.

To preliminary showcase the versatility of our PDE methodology in handling complex pipelines across diverse deployment platforms, we designed and implemented the advanced data-intensive pipelines of the UAERep project for execution on HPC infrastructure. The UAERep pipelines are dedicated to rainfall estimation and forecasting, leveraging a variety of heterogeneous data sources, including radar observations, satellite imagery, and precipitation gauge measurements, to generate accurate predictions. In this preliminary evaluation, for the sake of simplicity, the HPC environment is equipped with a specific configuration of Docker, Docker Compose, and the necessary libraries to support the execution of pipeline tasks by authorized

users in advance. The UAERep pipelines were specified using cWDL and executed through the PDE framework, resulting in two deployment recipes: one targeting our local experimental environment, and the other tailored for the UAERep HPC cluster. Although the two deployment recipes are largely similar, they differ in the data gathering phase. In the HPC cluster, data access is restricted to authorized users and confined to a designated directory, necessitating a specialized retrieval procedure. To accommodate these constraints, the data gathering tasks were polymorphically adapted within the PDE framework, ensuring full compliance with the access mechanism.

5.6 Chapter Conclusions

In this chapter, we introduced a polymorphic workflow composition methodology for choreographed workflows that dynamically adapts to specific security requirements and deployment contexts. We proposed PDE, an engine generating deployment recipes that preserve non-functional properties such as security and privacy. We evaluated PDE against existing workflow execution environments across both traditional cloud platforms and high-performance computing (HPC) clusters, demonstrating its ability to perform better in terms of complexity and remain infrastructure-independent.

The following chapter (Chapter 6) presents the final contribution of this thesis, that is, an RL-based approach to efficiently solve the placement problem in the Cloud-Edge Continuum. The approach builds upon the workflow and Continuum structure formalized in this work and provides a model to optimize several deployment-related costs, including resource consumption, property provision, and service acceptance.

6

Reinforcement Learning for Service Placement

This chapter presents a reinforcement learning methodology for service placement in the Cloud-Edge Continuum. The proposed approach formulates placement as a Markov Decision Process and leverages advanced DRL algorithms to optimize deployment under multiple objectives, including latency, resource utilization, and security. The methodology introduces a specialized training process and evaluates its effectiveness through extensive experimental analysis. This chapter pursues the effort, started in Chapter 5, to materialize an end-to-end deployment process and offers a practicable implementation of the matching function first formulated in Chapter 3.

6.1 Introduction

Service placement in Cloud-Edge Continuum is a highly complex problem. It involves assigning services to heterogeneous resources while balancing latency, cost, energy consumption, and security. While latency, cost and energy optimization have been deeply researched in the literature, security is the aspect that has received the least attention over the years. Despite that, critical applications require security properties, such as those comprised in the CIA triad (confidentiality, integrity, availability), to be enforced throughout the entire application life-cycle. Data processed by medical imaging or patient monitoring applications contain sensitive information, while autonomous vehicles could cause serious harm if data is tampered. In addi-

tion, the service placement problem is inherently NP-hard, and traditional optimization approaches, such as linear programming, heuristics, or metaheuristics, often fall short in dynamic, large-scale environments [95]. They either fail to adapt in real time or require prohibitive computational effort, limiting their applicability in practice.

The motivation for this chapter is the growing promise of reinforcement learning (RL) as a paradigm for placement optimization. RL is well-suited for environments characterized by uncertainty, variability, and high dimensionality, as it enables agents to learn effective policies through interaction with the system. Recent advances in deep reinforcement learning (DRL) further extend this potential, offering the ability to scale to complex problem spaces and integrate multi-objective optimization.

This chapter presents a reinforcement learning methodology for service placement in the Cloud-Edge Continuum. The methodology models placement as a Markov Decision Process (MDP), defines appropriate states, actions, and rewards, and employs a specialized DRL algorithm to optimize deployment. In addition, it integrates security-aware constraints, ensuring that the resulting policies are not only efficient but also robust against vulnerabilities.

The contributions of this chapter are threefold. First, we provide a formal MDP-based system model for service placement in heterogeneous environments. Second, we define an RL methodology and algorithmic design, including the use of Maskable PPO for efficient multi-objective learning. Third, we evaluate our methodology by comparing the proposed approach with heuristic and state-of-the-art solutions, highlighting its effectiveness in reducing missed deployments, overprovisioning, and resource variance.

6.1.1 Chapter Outline

The remainder of this chapter is organized as follows. Section 6.2 introduces the system model and deployment metrics. Section 6.3 presents our reinforcement learning methodology. Section 6.4 reports on the experimental evaluation. Finally, Section 6.5 concludes the chapter.

6.2 System Model

The Cloud–Edge Continuum represents a large-scale, heterogeneous, and dynamic environment in which services must be deployed while satisfying both functional and non-functional requirements. In our work, we focus on resources and security properties as the primary non-functional dimensions, leaving aside latency and vulnerability details that were considered in previous studies [17, 33]. The system model provides the formal basis for casting service placement as a reinforcement learning problem. It describes both the underlying infrastructure and the structure of applications, and defines a set of cost metrics that guide placement decisions.

6.2.1 Annotated Continuum Network

We begin by describing the infrastructure. The Continuum is naturally modeled as a graph structure capturing the facilities and their interconnections.

Definition 6.1 (Continuum network)**.** *A Continuum network is an annotated directed graph*

$$N_C = (F, L), \tag{6.1}$$

where $F = \{f_1, f_2, \dots, f_{|F|}\}$ is the finite set of facilities and $L \subseteq F \times F$ is the set of directed links between them.

Facilities can correspond to large-scale cloud data centers, regional edge servers, or small on-premise installations. This diversity reflects the actual deployment landscape, where high-capacity cloud nodes coexist with more resource-constrained edge nodes. Each facility $f \in F$ exposes two types of characteristics:

- **Resource capacities**. Quantitative descriptors of the available computational power, memory, and storage, expressed as a vector

$$C_f = (c_f^{cpu}, c_f^{mem}, c_f^{disk}). \tag{6.2}$$

  These values define the amount of CPU cores (c_f^{cpu}), memory (c_f^{mem}),

and storage space (c_f^{disk}) available to a facility and determine how many and what kind of services a facility can host concurrently.

- **Security properties**. Qualitative descriptors of the protections implemented by a facility, modeled as a subset

$$Sec_f \subseteq P, \tag{6.3}$$

where P is the set of all possible security properties considered in the system (e.g., confidentiality, integrity, compliance with a given standard). In our model, these properties are abstract, we are not concerned with the underlying mechanisms (encryption schemes, isolation techniques, etc.), but rather with their presence or absence.

This abstraction is crucial in the Cloud–Edge context, as different facilities may be operated by distinct providers and may expose only a subset of the expected security features. By modeling them as simple annotations, we allow service placement algorithms to reason explicitly about where security requirements can be met, rather than how they are implemented.

6.2.2 Annotated Application Workflow

The Continuum infrastructure provides resources to deploy applications. In the scope of this work, we consider applications structured as workflows of services.

Definition 6.2 (Application workflow)**.** *An application workflow is represented as an annotated directed graph*

$$W = (S, E), \tag{6.4}$$

where $S = \{s_1, s_2, \dots, s_{|S|}\}$ is the set of services and $E \subseteq S \times S$ is the set of directed edges capturing inter-service dependencies.

Each service $s \in S$ demands a defined amount of resources, determined by a demand vector

$$R_s = (r_s^{cpu}, r_s^{mem}, r_s^{disk}), \tag{6.5}$$

representing CPU cores (r_s^{cpu}), memory (r_s^{mem}), and storage space (r_s^{disk}) required. In addition, each service has a set of required security properties

$$Req_s \subseteq P, \tag{6.6}$$

defining the minimum set of properties a facility must have to host the service.

The graph structure of the workflow encodes ordering constraints: if $(s_i, s_j) \in E$, then service s_i must be deployed in a way that permits its outputs to reach s_j. In practice, this implies that dependencies must be placed on facilities connected by at least one path in N_C. Although our model abstracts communication delays, we maintain these constraints to ensure that workflows remain executable after placement.

A deployment is defined as a mapping function

$$M : S \to F, \tag{6.7}$$

which associates each service with a facility. This mapping must respect both the resource and security feasibility conditions described next.

6.2.3 Feasibility Constraints

The first requirement of a valid deployment is that services do not exceed the resources available on the chosen facilities. For every service $s \in S$ mapped to a facility $f = M(s)$, we must have

$$c_f^{cpu} \geq r_s^{cpu}, \quad c_f^{mem} \geq r_s^{mem}, \quad c_f^{disk} \geq r_s^{disk}. \tag{6.8}$$

The second requirement concerns security properties. A service can only be placed on a facility if all of its required properties are guaranteed by the facility. Formally, it is defined as

$$Req_s \subseteq Sec_f. \tag{6.9}$$

Definition 6.3 (Feasibility)**.** *A placement M is feasible if and only if, for every service $s \in S$, the conditions eq.* (6.8) *and eq.* (6.9) *hold.*

Feasibility ensures that services receive both the computational resources they require and the level of security demanded by their specification. If either condition fails, the deployment is invalid.

6.2.4 Deployment Metrics

Although feasibility is a necessary condition, it is far from sufficient to guarantee an effective placement. Different feasible mappings may lead to very different system behavior. For instance, one mapping may overload a single facility, another may consume scarce security-enabled facilities unnecessarily, and a third may fail to deploy some services altogether. To distinguish between such cases, we define four cost metrics that quantify deployment quality.

6.2.4.1 Deployment Cost

Deployment cost reflects the trade-off between consolidating services on few facilities and distributing them across many. Consolidation reduces the operational overhead of maintaining active nodes, while distribution improves resilience and may facilitate future deployments.

Definition 6.4 (Deployment cost)**.** *Let $U \subseteq F$ be the set of facilities actually used by a deployment. The deployment cost is defined as*

$$C_{dep} = \alpha \cdot \frac{|U|}{|F|} + (1 - \alpha) \cdot \left(1 - \frac{|U|}{|F|}\right), \tag{6.10}$$

where $\alpha \in [0, 1]$ controls the preference. When $\alpha \to 0$, consolidation is favored, minimizing the number of active facilities. When $\alpha \to 1$, the system encourages wider distribution.

6.2.4.2 Missed Deployment

Even if most services are placed, a deployment may fail to accommodate some due to resource exhaustion or missing properties. This reduces the effectiveness of the infrastructure and is penalized accordingly.

Definition 6.5 (Missed deployment)**.** *The missed deployment cost is defined as*

$$C_{miss} = \frac{1}{|S|} \sum_{s \in S} I_{miss}(s), \tag{6.11}$$

where $I_{miss}(s) = 1$ *if service* s *is not deployed and* 0 *otherwise. The metric ranges from* 0 *(all services placed) to* 1 *(no service placed).*

6.2.4.3 Deployment Imbalance

A deployment may be feasible but inefficient if some facilities are heavily loaded while others remain idle. This imbalance may degrade performance in practice and reduce robustness to failures. We capture this aspect using variance-based measures.

Definition 6.6 (Deployment imbalance)**.** *Let* $u_f^{cpu}, u_f^{mem}, u_f^{disk}$ *denote the utilization ratios of CPU, memory, and disk resources at facility* f*. The imbalance cost is*

$$C_{imb} = \frac{1}{3}\Big(Var(u^{cpu}) + Var(u^{mem}) + Var(u^{disk})\Big), \tag{6.12}$$

where the variance is computed across all facilities. Lower values indicate more balanced load distribution.

6.2.4.4 Overprovision

Overprovision occurs when a service is assigned to a facility that offers more security properties than the service requires. Although feasible, this wastes scarce secure facilities and can block the deployment of future services with stricter requirements.

Definition 6.7 (Overprovision)**.** *For a service* s *deployed on facility* f*, the overprovision is defined as*

$$I_{over}(s, f) = \begin{cases} 1 & \text{if } Req_s \subset Sec_f \text{ and } Req_s \neq Sec_f, \\ 0 & \text{otherwise.} \end{cases} \tag{6.13}$$

The overprovision cost is then

$$C_{over} = \frac{1}{|S|} \sum_{s \in S} I_{over}(s, M(s)). \tag{6.14}$$

This metric is particularly relevant in security-aware placement, as facilities offering strong security properties are typically fewer in number and should be preserved for services that explicitly require them.

6.2.4.5 Total Cost

Finally, we combine the four components into a single objective.

Definition 6.8 (Total cost)**.**

$$C_{tot} = w_{dep}C_{dep} + w_{miss}C_{miss} + w_{imb}C_{imb} + w_{over}C_{over}, \tag{6.15}$$

where weights $w_{dep}, w_{miss}, w_{imb}, w_{over} \geq 0$ are parameters. By adjusting them, system administrators or learning agents can emphasize different priorities, such as maximizing acceptance rate, balancing load, or conserving secure nodes.

6.3 Reinforcement Learning Methodology

The deployment metrics defined in Section 6.2 provide quantitative criteria for evaluating service placements. However, solving the placement problem through combinatorial optimization is infeasible at scale and under dynamic workloads. RL offers a way to derive adaptive strategies by interacting with a simulated environment: the RL agent observes the system, selects placement actions, and receives feedback in the form of rewards tied to deployment quality. Over time, the agent learns to balance conflicting objectives such as avoiding missed deployments, reducing imbalance, and minimizing overprovision.

To this end, we cast service placement as a Markov Decision Process following the framework illustrated in Figure 6.1, and formalize each component.

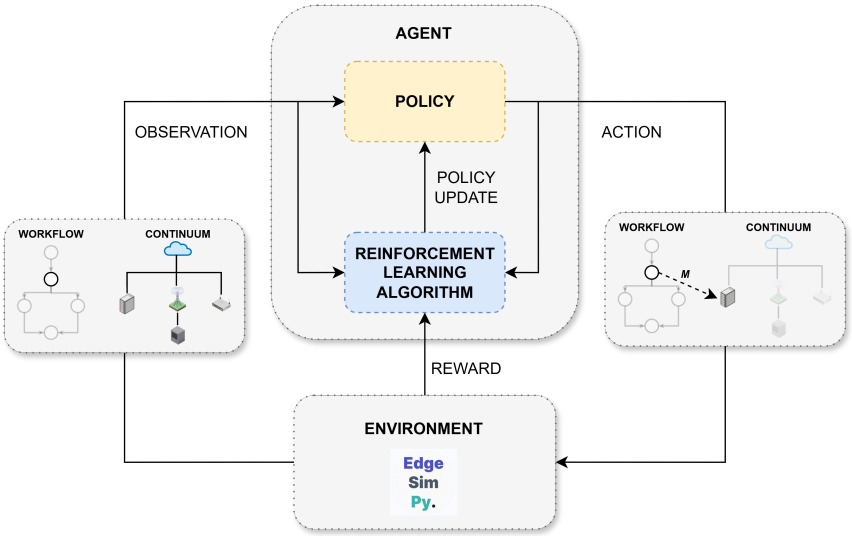

Figure 6.1: Learning framework that uses EdgeSimPy as Edge-Cloud Continuum simulator.

6.3.1 MDP Formulation

The service placement problem is defined as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$,

where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P}$the transition function, $\mathcal{R}$ the reward function, and $\gamma \in [0, 1]$ the discount factor.

6.3.2 Observations

The observation vector provides the agent with the most relevant information about the system state. It must include both the supply side (available resources and properties of facilities) and the demand side (requirements of the next service). In this way, the agent can compare what is offered with what is required, and make informed placement decisions.

At decision step t, the observation $\mathcal{o}_t$ is constructed with facility and service descriptors.

The facility descriptors represent, for each facility $f \in F$, the normalized remaining resources

$$res_f = \left(\frac{c_f^{cpu} - d_f^{cpu}}{c_f^{cpu}}, \frac{c_f^{mem} - d_f^{mem}}{c_f^{mem}}, \frac{c_f^{disk} - d_f^{disk}}{c_f^{disk}} \right), \tag{6.16}$$

where $d_f^{cpu}, d_f^{mem}, d_f^{disk}$ are the amount of used resources, alongside its binary vector of provided security properties $Sec_f \subseteq P$.

The service descriptor represents, for the current service $s \in S$, the normalized demand vector

$$R_s = \left(\frac{r_s^{cpu}}{\max_{s'} r_{s'}^{cpu}}, \frac{r_s^{mem}}{\max_{s'} r_{s'}^{mem}}, \frac{r_s^{disk}}{\max_{s'} r_{s'}^{disk}} \right), \tag{6.17}$$

together with its property requirements vector corresponding to $Req_s \subseteq P$.

The full state vector is the concatenation of all facility descriptors and the current service descriptor as

$$\mathcal{s}_t = [res_{f_1}, Sec_{f_1}, \dots, res_{f_{|F|}}, Sec_{f_{|F|}}, R_s, Req_s]. \tag{6.18}$$

This design ensures that the agent perceives both infrastructure status and request requirements, making it possible to learn placements that balance supply and demand.

6.3.3 Actions

The actions correspond to feasible decisions: assigning the service to a specific facility, or rejecting it if no facility is suitable. This reflects the real-world orchestration task and allows the agent to explicitly account for missed deployments. The discrete action space is

$$\mathcal{A} = F \cup \{a_{reject}\}. \tag{6.19}$$

Each action either maps the current service to a facility $f \in F$ or rejects it with a_{reject}. Infeasible actions (e.g., insufficient capacity or missing properties) are excluded through action masking, which acts as a filter to avoid unwanted results. This prevents the agent from exploring meaningless or

invalid options, improving sample efficiency and stability.

6.3.4 Transitions

Transitions describe how the system evolves after a placement decision: resource capacities decrease, imbalance may increase, and overprovision may occur. Capturing these dynamics is essential, since decisions taken at time t dictate feasibility at time $t + 1$. The transition function P deterministically updates the environment given state $\mathcal{s}_t$ and action $\mathcal{a}_t$. If the service is deployed on facility f, the facility's resources are reduced and the system metrics (deployment cost, imbalance, overprovision) are updated accordingly. If the service is rejected, it contributes to the missed deployment cost. In either case, the next state $\mathcal{s}_{t+1}$ corresponds to the updated infrastructure and the next pending service.

6.3.5 Rewards

The reward integrates the four deployment costs. It is the central driver of learning, guiding the agent to minimize undesirable outcomes (missed deployments, imbalance, etc.) while still respecting feasibility. At each decision step, the agent receives an immediate reward equal to the negative weighted sum of costs

$$\mathcal{r}_t = -\Big(w_{dep}C_{dep}(t) + w_{miss}C_{miss}(t) + w_{imb}C_{imb}(t) + w_{over}C_{over}(t)\Big). \tag{6.20}$$

This formulation aligns the agent's learning objective with the placement optimization goal, that is, maximizing long-term reward and, correspondingly, minimizing long-term deployment costs.

6.3.6 Episodes

Finally, episodes are structured as workflows. This allows the agent to learn from sequences of decisions, rather than isolated placements, encouraging strategies that remain effective across the lifetime of an application. An episode corresponds to the deployment of an application workflow. It starts

with an empty mapping and ends when all services are processed or when a maximum step limit is reached. The cumulative reward of an episode measures the quality of the overall deployment.

6.3.7 Algorithm Choice

We adopt Proximal Policy Optimization with action masking. PPO is a policy-gradient algorithm known for its robustness and stability. The clipped objective prevents destructive updates, a common issue in dynamic and high-dimensional environments like ours. Action masking is essential, as it avoids infeasible actions and accelerates exploration.

Standard PPO is a policy gradient algorithm. It improves the policy π_θ by iteratively sampling trajectories, estimating advantages, and applying gradient ascent on a clipped surrogate objective. This objective prevents overly large updates that could destabilize training:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min\big(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big)\Big], \tag{6.21}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between new and old policies, $\hat{A}_t$ is the advantage estimate, and ϵ is the clipping parameter. This ensures updates remain within a "trust region" around the previous policy, combining stability with efficiency.

However, applying standard PPO to service placement faces a major issue, as the action space contains many infeasible actions. For example, a service may request confidentiality and integrity, but many facilities lack one of these properties. Exploring those infeasible actions is wasteful, and penalizing them post-hoc (by giving negative rewards) slows down learning. Maskable PPO extends PPO by introducing action masks directly into the policy sampling step.

At each state s, the mask is a Boolean vector $m(s) \in \{0,1\}^{|\mathcal{A}|}$, where $m(s)_i = 1$ if action a_i is feasible and $m(s)_i = 0$ otherwise.

Instead of sampling from the full distribution $\pi_\theta(\cdot|s)$, the algorithm samples from a masked distribution:

$$\pi_\theta^M(a|s) = \frac{\pi_\theta(a|s) \cdot m(s)_a}{\sum_{a' \in \mathcal{A}} \pi_\theta(a'|s) \cdot m(s)_{a'}}. \tag{6.22}$$

In practice, this means the logits of infeasible actions are set to $-\infty$ before applying the softmax, ensuring they have zero probability. This modification preserves PPO's clipped objective but ensures that all updates are conditioned only on feasible actions. The surrogate loss is unchanged in form, but the action distribution is masked:

$$L^{CLIP-MASK}(\theta) = \hat{\mathbb{E}}_t\Big[\min\left(r_t^M(\theta)\hat{A}_t,\ \text{clip}(r_t^M(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\Big], \tag{6.23}$$

with $r_t^M(\theta) = \frac{\pi_\theta^M(a_t|s_t)}{\pi_{\theta_{old}}^M(a_t|s_t)}$.

The advantages of Maskable PPO are its sample efficiency, as exploration is focused exclusively on valid actions, accelerating policy improvement, and its reduced variance, since the agent no longer receives large negative rewards for infeasible attempts, which often dominate the reward signal. Additionally, even early in training, the agent respects hard feasibility constraints, avoiding misleading results and enhancing safety.

In our context, this is critical. The proportion of infeasible actions grows with the number of facilities and properties. Without masking, the agent would waste most of its rollouts on actions that cannot succeed. With Maskable PPO, the effective action space is reduced dynamically at every step, making the algorithm scalable.

6.4 Experimental Evaluation

In this section, we present the experimental evaluation of our RL-based model for service placement in the Cloud–Edge Continuum. The analysis focuses on three main aspects: i) optimization of resource utilization balance, ii) performance with respect to property overprovisioning and missed deployments, and iii) computation time of the placement process.

For training and evaluation, we employed EdgeSimPy, an Edge–Cloud simulator that supports the deployment of containerized, service-based ap-

plications across a network of nodes. The simulator allows specifying the characteristics of the nodes (e.g., CPU, memory, and storage) as well as the requirements of the services to be deployed (e.g., CPU, memory, and storage demands). We extended the simulator to incorporate security requirements and capabilities. The resulting environment is then wrapped to ensure compatibility with the Gym interface.

To benchmark the model's performance, we compared it against the following baseline strategies:

- *Stochastic (ST) strategy*: randomly selects a node among those available.
- *Greedy strategies*: pool of approaches that evaluate all possible placement options and selects the one yielding the lowest reward according to a give metric, including CPU usage (*Greedy-CPU*), memory usage (*Greedy-Mem*), available storage space (*Greedy-Disk*), resource usage imbalance (*Greedy-Imb*), amount of overprovisioned services (*Greedy-OP*), and deployment cost (*Greedy-DC*).

6.4.1 Experimental Setup

All experiments were conducted on a machine running Ubuntu 24.10, equipped with an Intel ® Core i7 12850HK 16-Core (24-thread) CPUs @ 4.80 GHz, a Nvidia RTX A3000® GPU, and 32 GB of RAM. The algorithms were implemented in Python 3.12.3, with training carried out using RL Baselines3 (v2.6.1).

Each experimental scenario comprises 500 services distributed across 50 applications (10 services per application) and deployed over 7 infrastructure nodes. At the beginning of each simulation, node characteristics and service demands (CPU, memory, disk space, and security properties) are randomized. The parameters used in the simulation setup are listed in Table 6.1.

6.4.2 Missed Deployments

This experiment evaluates the number of services that remain undeployed at the end of an episode (500 steps). The metric reflects both the efficiency

Table 6.1: Parameter settings.

Parameter	Symbol	Value
Number of services	S	500
Number of workflows	W	10
Number of facilities	F	7
Number of properties	P	2
CPU demand	r_s^{cpu}	[1, 4]
Memory demand	r_s^{mem}	{2, 4} GB
Storage demand	r_s^{disk}	[20, 30] GB
CPU capacity	c_f^{cpu}	[190, 200]
Memory capacity	c_f^{mem}	{256, 521} GB
Disk capacity	c_f^{disk}	{2, 4} TB
Learning rate	α_{lr}	$8.8 \cdot 10^{-5}$
Discount factor	γ	0.96
Update frequency	N_{copy}	512 iterations

of the resource distribution and the effectiveness of the deployment strategy over time.

As shown in Figure 6.2, across 100 simulations with 500 services each, ST is the strategy that performs considerably worse compared to the others. Among greedy strategies, Greedy-Disk and Greedy-DC reach a higher amount of missed deployments, showing that they are not sufficient alone to optimize the number of accepted services. Our strategy outperforms all the benchmark strategies, exhibiting good performance in keeping the number of rejected services low. Among the greedy strategies, Greedy-OP achieves the lowest number of missed deployments, confirming that avoiding overprovisions can increase the number of accepted services in our system setting. The results of all experiments are reported in Table 6.2.

6.4.3 Overprovisioning

This experiment assesses the model's ability to minimize overprovisioning, i.e., assigning services to nodes offering more properties than re-

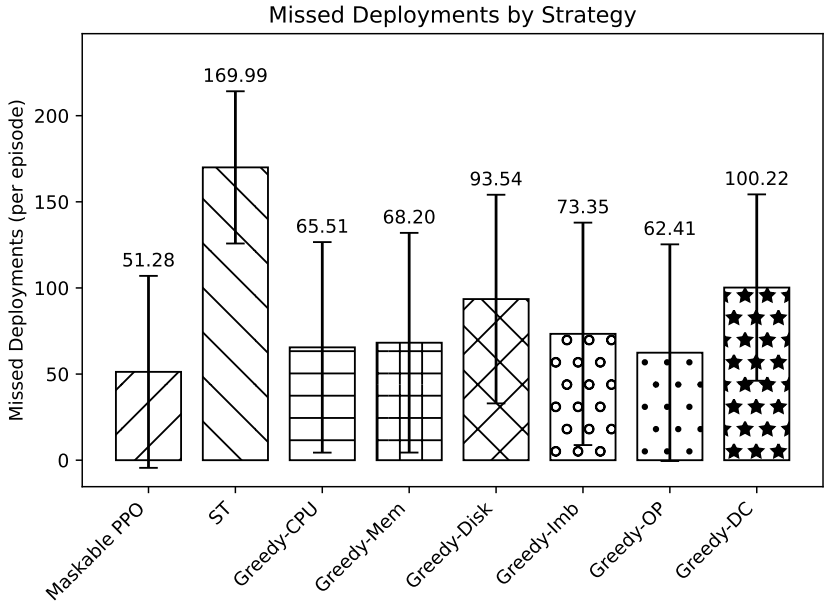

Figure 6.2: Average number of missed deployments over 100 simulations with 500 services each.

quired. Avoiding overprovisioning is desirable, as it preserves the capacity of resource-rich nodes for future service placements.

Over 100 simulations with 500 services each, Maskable PPO and greedy strategies yield similar results (shown in Figure 6.3). In this evaluation, ST performs better that the other approaches, although this result stems from the fact that more services remain undeployed. As fewer services are successfully deployed with the ST strategy, there are fewer services that can be overprovisioned. The same argument can be applied to Greedy-Disk and Greedy-DC, that show poorer performances in the missed deployments evaluation.

Table 6.2: Comparison of MaskablePPO, Stochastic, and Greedy strategies across evaluation metrics (mean ± std over 100 simulations).

Approach	Missed Deployments	Overprovision	Variance
Maskable PPO	51.28 ± 55.72	337.53 ± 42.44	0.05 ± 0.01
Stochastic	169.99 ± 44.19	247.64 ± 34.50	0.05 ± 0.02
Greedy-CPU	65.51 ± 61.11	325.61 ± 46.35	0.03 ± 0.01
Greedy-Mem	68.2 ± 63.77	323.37 ± 47.92	0.04 ± 0.01
Greedy-Disk	93.54 ± 60.56	303.54 ± 46.02	0.07 ± 0.02
Greedy-Imb	73.35 ± 64.56	319.4 ± 48.27	0.05 ± 0.01
Greedy-OP	62.41 ± 62.90	328.03 ± 47.50	0.08 ± 0.04
Greedy-DC	100.22 ± 54.09	298.22 ± 41.20	0.1 ± 0.03

6.4.4 Resource Distribution Variance

This experiment investigates how evenly the model distributes resource demand across nodes, considering CPU, memory, and storage utilization. A lower variance across nodes helps prevent early saturation, thereby preserving resources for later stages of the simulation and enabling the placement of more services.

Results are shown in Figure 6.4. After 100 simulations with 500 services each, our approach is outperformed only by Greedy-CPU and Greedy-Mem. Greedy-Imb, which places services with the aim to use the computional resources of facilities as evenly as possible, starts the simulations with the lowest variance, but gets outperformed after ~150 services by the three best strategies. ST, performing a uniformly random placement, shows a low final variance with an approximately linear trend. Conversely, Greedy-OP and Greedy-DC show the worse behavior with a distinct peak at ~300 services followed by a descending trend resulting from the saturation of the resources.

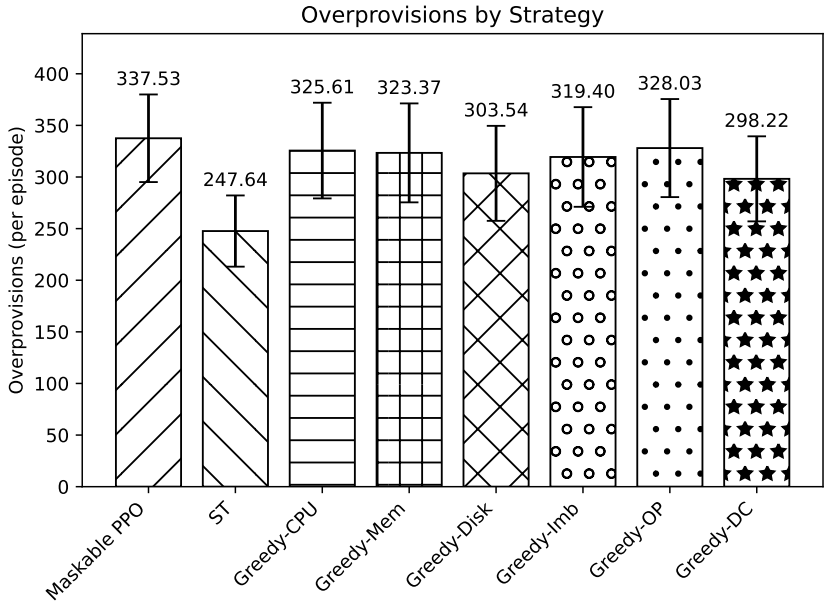

Figure 6.3: Average number of overprovisioned services across 200 simulations with 100 services each.

6.5 Conclusion

This work presented a reinforcement learning approach for security-aware service placement in the Cloud–Edge Continuum. By modeling the problem as a Markov Decision Process and employing Maskable PPO, we developed a placement strategy that accounts for resource heterogeneity and security requirements while minimizing deployment cost, missed deployments, imbalance, and overprovisioning.

Experiments on an extended version of EdgeSimPy show that the proposed method performs competitively, reducing constraint violations and achieving results close to exhaustive search while outperforming stochastic baselines.

The following chapter (Chapter 7) draws the conclusions of the thesis.

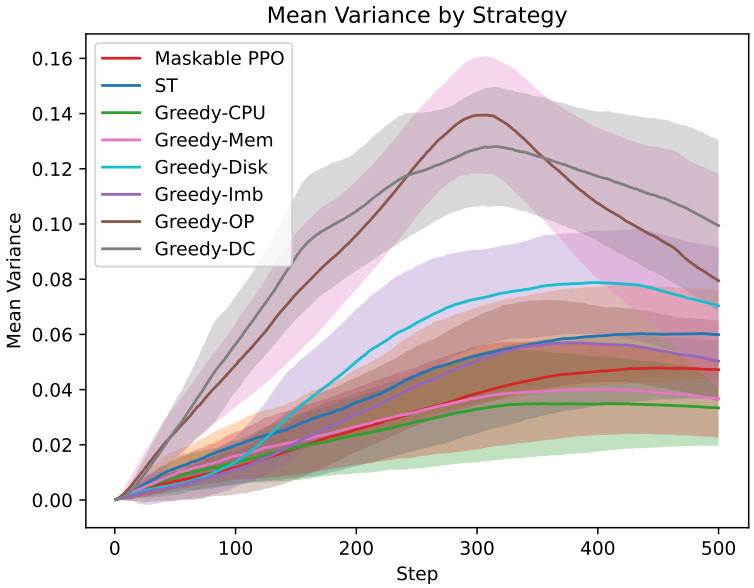

Figure 6.4: Average variance of CPU, memory, and storage utilization across 100 simulations with 500 services each.

7 Conclusions

In this thesis, we investigated the challenges of service composition and placement in the Cloud-Edge Continuum, with a focus on data-intensive and security-sensitive applications. The thesis aimed to bridge the gap between the promise of continuum computing and the limitations of current deployment strategies, languages, and optimization methods.

After introducing the motivations and objectives (Chapter 1), we reviewed related work in service placement, workflow description languages, and reinforcement learning approaches (Chapter 2). We then proposed a QoS-aware deployment framework (Chapter 3), a vulnerability-aware placement methodology (Chapter 4), a polymorphic workflow description language (Chapter 5), and a reinforcement learning approach for placement optimization (Chapter 6). Each of these contributions was validated experimentally, demonstrating their feasibility and effectiveness.

The following sections summarize the contributions (Section 7.1) and outline open challenges and directions for future work (Section 7.2).

7.1 Summary of the Contributions

The contributions of this thesis are manifold and address key challenges in service composition and placement.

QoS-Aware Deployment Framework. We presented a novel framework for deploying service compositions in the Cloud-Edge Continuum. The framework models workflows and facilities using annotated graphs and

performs requirement-resource matching to generate deployment recipes. This property-driven approach enables developers to deploy applications across heterogeneous infrastructures while preserving non-functional requirements such as latency, integrity, and confidentiality.

Vulnerability-Aware Placement and Migration. We proposed a methodology that integrates vulnerability assessment directly into placement decisions. By modeling vulnerabilities and their impact on service stability, the approach accounts for migration costs and introduces mechanisms to mitigate recursive migrations. This improves both the security posture and the efficiency of deployments in dynamic environments.

Polymorphic Workflow Description Language (cWDL). We designed a workflow description language tailored for the continuum, extending existing models with annotations, choreography constructs, and polymorphism. The language is supported by a deployment engine capable of interpreting annotated workflows and generating polymorphic deployment artifacts. This bridges the gap between high-level workflow design and executable deployments, enhancing flexibility and runtime adaptability.

Reinforcement Learning for Service Placement. We formulated service placement as a Markov Decision Process and introduced a reinforcement learning methodology leveraging Maskable PPO. This contribution includes the formalization of system states, actions, and rewards, as well as the integration of security constraints into the training process. Experimental evaluation shows that our RL-based approach significantly reduces missed deployments, overprovisioning, and variance compared to heuristic and baseline methods.

Together, these contributions advance the state of the art in continuum computing by offering theoretical models, practical tools, and validated methodologies for secure, efficient, and intelligent service composition and placement.

7.2 Future Work

The research presented in this thesis opens several directions for future investigation.

Integration of Multiple Objectives. While our RL methodology accounts for latency, resource efficiency, and security, future work should extend it to consider energy efficiency, carbon footprint, and cost-awareness. These objectives are increasingly important for sustainable computing and green AI.

Trust and Decentralization. The proposed frameworks rely on system-level models and trusted placement decisions. Future research could explore decentralized orchestration mechanisms, leveraging blockchain or distributed consensus to enhance transparency and trust in continuum deployments.

Compositional Security Guarantees. Extending the vulnerability-aware approach to composite workflows remains an open challenge. Certifying the security of entire service compositions based on the properties of individual services and facilities would significantly enhance trustworthiness.

Adaptive and Self-Evolving Workflows. The proposed cWDL supports polymorphism and annotations, but future work could extend it with self-adaptive mechanisms, where workflows evolve at runtime by learning from past deployments and adapting to changing conditions.

Large-Scale Real-World Validation. While this thesis includes experimental evaluations, future work should focus on large-scale industrial pilots, integrating our methodologies into real ecosystems such as 5G-enabled smart cities, healthcare infrastructures, or critical manufacturing pipelines.

Hybrid Learning Approaches. The reinforcement learning methodology could be combined with traditional optimization techniques or metaheuristics in hybrid frameworks, exploiting the strengths of both approaches for

faster convergence and higher quality decisions.

7.3 Closing Remarks

This thesis has proposed frameworks, languages, and methodologies that collectively enable secure, QoS-aware, and intelligent service deployment in the Cloud-Edge Continuum. By addressing fundamental challenges in placement, workflow description, and optimization, the thesis contributes to making continuum computing a viable paradigm for future distributed systems.

The work presented here constitutes a step toward trustworthy and efficient service orchestration across heterogeneous infrastructures, but it also highlights the vast research opportunities that remain. Bridging the gap between theoretical models and large-scale industrial adoption will be the key to unlocking the full potential of the Cloud-Edge Continuum.

Appendices

A

Publications

This work has appeared in varying forms, in the form of journals and conference papers. In the following we report papers and articles published or submitted used in this thesis.

P1: QoS-Aware Deployment of Service Compositions in 5G-Empowered Edge-Cloud Continuum

Co-authors: M. Anisetti, F. Berto

In: Proc. of 2023 IEEE 16th International Conference on Cloud Computing (CLOUD), July 2023

Abstract: Nowadays, modern service compositions are increasingly adopted in critical scenarios where advanced Quality of Services (QoS) such as low latency, security, and privacy are fundamental. The landing platforms for the deployment of such compositions are progressively becoming capable to offer capabil-ities that support such advanced QoS requests (e.g., low latency via 5G network slice) spanning the Edge-Cloud Continuum. Actual deployment solutions focus mainly on resource allocation (i.e., CPU, memory, and storage), falling short of addressing advanced QoS and unleashing the true potential of the Edge-Cloud Continuum. In this paper, we present an automatic QoS-aware deployment solution for composed services in the Edge-Cloud Continuum. It compares QoS requests on the service composition with the capabilities of a given continuum in order to find, generate and execute suitable deployment recipes. Our preliminary experimental evaluation demonstrates the feasibility of our solution in

a realistic scenario.

P2: Advanced IoT Edge Architecture for Smart City

Co-authors: M. Anisetti, D. Cazzetta

In: Proc. of 2023 17th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS), Nov. 2023

Abstract: In the last few years, terms like Internet of Things, Cloud Computing and Edge Computing have captured a lot of attention from both scientific and industrial perspectives. They are independent but closely related concepts, and their use extends into several scenarios, such as Smart Agriculture, Smart Home, autonomous vehicles, and Smart City, to name but a few. Among the Smart City solutions, Smart Parking aims to solve the problem of parking space and parking management in big cities, since the search for free parking spaces brings along a serious cost, air pollution and stress issues. The objective of this article is to propose an Edge Computing architecture tailored for Smart Parking solutions, addressing the challenges of real-time data processing and efficient parking management. This architecture integrates a Deep Learning solution based on Gated Recurrent Units (GRU) to accurately forecast the number of available and occupied parking spaces.

P3: MUSA: A Platform for Data-Intensive Services in Edge-Cloud Continuum

Co-authors: M. Anisetti, C. A. Ardagna, M. Banzi, F. Berto, E. Damiani, A. Pedretti, A. Pisati, A. Retico

In: Lecture Notes on Data Engineering and Communications Technologies, vol. 203.

Abstract: In the rapidly evolving landscape of modern applications, the Edge-Cloud Continuum emerges as a pivotal paradigm, promising unprecedented flexibility and efficiency in service deployments. As the demand for low-latency, high-throughput applications intensifies, the Continuum provides a dynamic framework, enabling the distribution of computational tasks between centralized cloud servers and decentralized edge devices. However, the transition from traditional

cloud-centric models to the Continuum introduces complexities that necessitate careful consideration. Besides development, Continuum applications call for a placement process with the aim to allocate services to the best suitable deployment node, according to application requirements and nodes capabilities. Furthermore, controlling non-functional properties within the Cloud-Edge Continuum and balancing trade-offs between performance, reliability, and security becomes increasingly intricate in this distributed architecture. This paper addresses the above challenges proposing MUSA, a deployment platform for data-intensive workflows of services integrating continuous non-functional properties verification.

P4: Vulnerability-Aware Secure Service Deployment in Cloud-Edge Continuum

Co-authors: M. Anisetti, C. A. Ardagna, R. Badonnel, N. Schnepf
In: IEEE Transactions on Network and Service Management, 2025

Abstract: Software weaknesses and vulnerabilities are continuously discovered and rapidly evolving. Their direct and indirect interference with the business process workflow execution is neither fully understood nor addressed by the current literature. The strict control of the vulnerability footprint of the landing platform before cloud/web service workflow execution is nowadays largely used as a prevention measure in order to improve execution trustworthiness. The vulnerability footprint governance is exacerbated by the cloud, where a common execution platform hosting (vulnerable) services is shared between different tenants. The paper proposes a service workflow deployment solution tailored for Edge-Cloud Continuum, made of different landing platforms showing different peculiarities. The proposed solution is capable of finding a suitable deployment recipe for a given workflow by i) evaluating the vulnerability footprint of each platform, ii) computing the set of candidate deployment platforms, iii) finding the optimal deployment solution, and iv) migrating already deployed workflows in case the vulnerability requirement is no longer satisfied. Each workflow can be associated with a set of requirements to be satisfied by our deployment solution, like the maximum level of vulnerability

footprint accepted. Each workflow deployment contributes to the vulnerability footprint of the landing platform involved.

P5: Secure and Polymorphic Composition of Service Workflows in the Cloud Continuum

Co-authors: M. Anisetti, E. Damiani, M. Gala, A. Ovena, M. Perfumo

To appear in proceedings of: 17th International Conference on Management of Digital Ecosystems, 2025

Abstract: Modern applications are increasingly built as compositions of services deployed across diverse platforms. Ensuring the security and privacy of these service compositions remains a significant challenge, particularly within the dynamic and distributed environment of the Cloud Continuum. In such contexts, choreography, which avoids reliance on a centralized orchestrator, is often more suitable than traditional orchestration approaches. In this paper, we introduce a polymorphic workflow composition methodology that supports the design of choreographed workflows capable of adapting their structure (i.e., the polymorphism) and behavior based on specific security requirements and deployment contexts. Our approach generates deployment recipes that preserve critical non-functional properties, such as security and privacy. We evaluate our methodology in both traditional Cloud Continuum environments and high-performance computing (HPC) clusters, demonstrating its ability to maintain deployment independence from the underlying infrastructure.

Acronyms

A3C Asynchronous Advantage Actor-Critic.

AES Advanced Encryption Standard.

AI Artificial Intelligence.

BPEL Business Process Execution Language.

BPMN Business Process Model and Notation.

CAWL Context-Aware Workflow Language.

CDC Coded Distributed Computation.

CSP Cloud Service Provider.

CVE Common Vulnerabilities and Exposures.

CVSS Common Vulnerability Scoring System.

cWDL choreographic Workflow Description Language.

DAG Directed Acyclic Graph.

DDPG Deep Deterministic Policy Gradient.

DG-DDPG Dual-GNN Deep Deterministic Policy Gradient.

DQN Deep Q-Network.

DRL Deep Reinforcement Learning.

ECC Edge-Cloud Continuum.

ESPP Edge Server Placement Problem.

FaaS Function as a Service.

FC Fog Computing.

FL Federated Learning.

GCN Graph Convolutional Network.

GNN Graph Neural Network.

HPC High Performance Computing.

IDS Intrusion Detection System.

IoT Internet of Things.

IPS Intrusion Prevention System.

JCCSP Joint Caching and Computing Service Placement.

LP Linear Programming.

LSTM Long Short-Term Memory.

MARL Multiagent Reinforcement Learning.

MDP Markov Decision Process.

MEC Mobile Edge Computing.

NLP Nonlinear Programming.

NoM Number of Migrations.

NVD National Vulnerability Database.

OVAL Open Vulnerability and Assessment Language.

PDE Polymorphic Deployment Engine.

PDQN Parameterized Deep Q-Network.

POWL Partially Ordered Workflow Language.

PPO Proximal Policy Optimization.

QoE Quality of Experience.

QoS Quality of Service.

RL Reinforcement Learning.

SAR Service Acceptance Ratio.

SFC Service Function Chain.

SFCP Service Function Chain Placement.

SGE Service Graph Embedding.

SL Split Learning.

TD Twin-Delayed.

TDC Total Deployment Cost.

VM Virtual Machine.

VNF Virtual Network Function.

WATQD Weight-Averaged Twin-Q-Delayed.

WDL Workflow Description Language.

WS-CDL Web Services Choreography Description Language.

YAWL Yet Another Workflow Language.

Bibliography

[1] M. Aazam, S. Zeadally, and K. A. Harras. "Fog Computing Architecture, Evaluation, and Future Research Directions".
In: *IEEE Communications Magazine* 56.5 (May 2018), pp. 46–52.
DOI: 10.1109/MCOM.2018.1700707.

[2] M. Abbas, S. Khan, A. Monum, F. Zaffar, R. Tahir, D. Eyers, H. Irshad, A. Gehani, V. Yegneswaran, and T. Pasquier.
"PACED: Provenance-based Automated Container Escape Detection".
In: *2022 IEEE International Conference on Cloud Engineering (IC2E)*. 2022, pp. 261–272. DOI: 10.1109/IC2E55432.2022.00035.

[3] S. Abdulmalek, A. Nasir, W. A. Jabbar, M. A. M. Almuhaya, A. K. Bairagi, M. A.-M. Khan, and S.-H. Kee. "IoT-Based Healthcare-Monitoring System towards Improving Quality of Life: A Review".
In: *Healthcare* 10.10 (Oct. 2022), p. 1993.
DOI: 10.3390/healthcare10101993.

[4] V. Afzali Gorooh, M. Ghazvinian, W. Hu, Q. Cao, M. Pan, B. Hassan Banimfreg, E. Damiani, A. Sengupta, D. Axisa, and L. Delle Monache. "High Spatiotemporal Resolution Quantitative Precipitation Estimation over the United Arab Emirates".
In: *104th Annual AMS Meeting 2024*. Vol. 104. 2024, p. 429038.

[5] F. Ait Salaht, F. Desprez, A. Lebre, C. Prud'homme, and M. Abderrahim. "Service Placement in Fog Computing Using Constraint Programming".
In: *2019 IEEE International Conference on Services Computing (SCC)*. July 2019, pp. 19–27. DOI: 10.1109/SCC.2019.00017.

[6] N. Akhtar, A. Raza, V. Ishakian, and I. Matta.
"COSE: Configuring Serverless Functions using Statistical Learning".
In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. ISSN: 2641-9874. July 2020, pp. 129–138.

[7] I. Alghamdi, C. Anagnostopoulos, and D. P. Pezaros.
"Delay-Tolerant Sequential Decision Making for Task Offloading in Mobile Edge Computing Environments". In: *Information* 10.10 (Oct. 2019), p. 312.
DOI: 10.3390/info10100312.

[8] B. Ali, M. A. Gregory, and S. Li. “Multi-Access Edge Computing Architecture, Data Security and Privacy: A Review”. In: *IEEE Access* 9 (2021), pp. 18706–18721. DOI: 10.1109/ACCESS.2021.3053233.

[9] S. Alrabaee, M. Debbabi, P. Shirani, L. Wang, A. Youssef, A. Rahimian, L. Nouh, D. Mouheb, H. Huang, and A. Hanna. “Binary Code Fingerprinting for Cybersecurity: Application to Malicious Code Fingerprinting”. English. In: *Advances in Information Security*. Advances in Information Security. Springer, 2020, pp. 1–249. DOI: 10.1007/978-3-030-34238-8_1.

[10] D. Alsadie. “A Comprehensive Review of AI Techniques for Resource Management in Fog Computing: Trends, Challenges, and Future Directions”. In: *IEEE Access* 12 (2024), pp. 118007–118059. DOI: 10.1109/ACCESS.2024.3447097.

[11] M. Anisetti, C. A. Ardagna, F. Berto, and E. Damiani. “A Security Certification Scheme for Information-Centric Networks”. In: *IEEE Transactions on Network and Service Management* 19.3 (2022), pp. 2397–2408.

[12] M. Anisetti, C. A. Ardagna, M. Banzi, F. Berto, R. Bondaruc, E. Damiani, A. Pedretti, A. Pisati, and A. Retico. “MUSA: A Platform for Data-Intensive Services in Edge-Cloud Continuum”. In: *Advanced Information Networking and Applications*. Ed. by L. Barolli. Cham: Springer Nature Switzerland, 2024, pp. 327–337. DOI: 10.1007/978-3-031-57931-8_32.

[13] M. Anisetti, C. A. Ardagna, and N. Bena. “Multi-Dimensional Certification of Modern Distributed Systems”. In: *IEEE Transactions on Services Computing* (2022), pp. 1–14.

[14] M. Anisetti, C. A. Ardagna, N. Bena, and E. Damiani. “An Assurance Framework and Process for Hybrid Systems”. In: *Communications in Computer and Information Science*. Vol. 1484 CCIS. 2021, pp. 79–101.

[15] M. Anisetti, C. A. Ardagna, and E. Damiani. “Security Certification of Composite Services: A Test- Based Approach”. In: *2013 IEEE 20th International Conference on Web Services*. June 2013, pp. 475–482.

[16] M. Anisetti, C. A. Ardagna, E. Damiani, F. Gaudenzi, and G. Jeon. "Cost-effective deployment of certified cloud composite services". In: *Journal of Parallel and Distributed Computing* 135 (2020), pp. 203–218.

[17] M. Anisetti, F. Berto, and R. Bondaruc. "QoS-aware Deployment of Service Compositions in 5G-empowered Edge-Cloud Continuum". In: *2023 IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 2023, pp. 471–478.

[18] C. A. Ardagna, V. Bellandi, M. Bezzi, P. Ceravolo, E. Damiani, and C. Hebert. "Model-Based Big Data Analytics-as-a-Service: Take Big Data to the Next Level". In: *IEEE Transactions on Services Computing* 14.2 (2021), pp. 516–529.

[19] R. G. Aryal and J. Altmann. "Dynamic Application Deployment in Federations of Clouds and Edge Resources Using a Multiobjective Optimization AI Algorithm". In: *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. Apr. 2018, pp. 147–154. DOI: 10.1109/FMEC.2018.8364057.

[20] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou. "On Uncoordinated Service Placement in Edge-Clouds". In: *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Dec. 2017, pp. 41–48. DOI: 10.1109/CloudCom.2017.46.

[21] M. Autili, A. Di Salle, F. Gallo, C. Pompilio, and M. Tivoli. "CHOReVOLUTION: Service Choreography in Practice". In: *Science of Computer Programming* 197 (Oct. 2020), p. 102498. DOI: 10.1016/j.scico.2020.102498.

[22] C. Avasalcai, C. Tsigkanos, and S. Dustdar. "Decentralized Resource Auctioning for Latency-Sensitive Edge Computing". In: *2019 IEEE International Conference on Edge Computing (EDGE)*. July 2019, pp. 72–76. DOI: 10.1109/EDGE.2019.00027.

[23] T. Bahreini and D. Grosu. "Efficient Placement of Multi-Component Applications in Edge Computing Systems". In: *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. SEC '17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1–11. DOI: 10.1145/3132211.3134454.

[24] J. Banghart and C. Johnson. *The Technical Specification for the Security Content Automation Protocol (SCAP). NIST Special Publication*. 2011.

[25] B. Banimfreg, E. Damiani, V. A. Gorooh, D. Axisa, L. D. Monache, and Y. Wehbe. “Innovative approach for gauge-based QPE in arid climates: comparing neural networks and traditional methods”. In: *Journal of Big Data* 12.1 (2025), pp. 1–36.

[26] M. Barrère, R. Badonnel, and O. Festor. “Towards the Assessment of Distributed Vulnerabilities in Autonomic Networks and Systems”. In: *Proc. of the IEEE/IFIP Network Operations and Management Symposium (IEEE/IFIP NOMS)*. Apr. 2012, pp. 335–342. DOI: 10.1109/NOMS.2012.6211916.

[27] M. Barrère. “Vulnerability Management for Safe Configurations in Autonomic Networks and Systems”. Phd Thesis. University of Lorraine, France, 2016. URL: http://www.theses.fr/2014LORR0048.

[28] M. Barrére, R. Badonnel, and O. Festor. “A SAT-based Autonomous Strategy for Security Vulnerability Management”. In: *2014 IEEE Network Operations and Management Symposium (NOMS)*. 2014, pp. 1–9. DOI: 10.1109/NOMS.2014.6838309.

[29] L. Belcastro, F. Marozzo, A. Orsino, D. Talia, and P. Trunfio. *Navigating the Edge-Cloud Continuum: A State-of-Practice Survey*. May 2025. DOI: 10.48550/arXiv.2506.02003. arXiv: 2506.02003 [cs].

[30] A. Betzler, C. Gomez, I. Demirkol, and J. Paradells. “CoCoA+: An Advanced Congestion Control Mechanism for CoAP”. In: *Ad Hoc Networks* 33 (Oct. 2015), pp. 126–139. DOI: 10.1016/j.adhoc.2015.04.007.

[31] Y. Bi, C. C. Meixner, M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou. “Multi-Objective Deep Reinforcement Learning Assisted Service Function Chains Placement”. In: *IEEE Transactions on Network and Service Management* 18.4 (Dec. 2021), pp. 4134–4150. DOI: 10.1109/TNSM.2021.3127685.

[32] A. Bocci, S. Forti, G.-L. Ferrari, and A. Brogi. “Declarative Secure Placement of FaaS Orchestrations in the Cloud-Edge Continuum”. en. In: *Electronics* 12.6 (Jan. 2023), p. 1332. ISSN: 2079-9292. DOI: 10.3390/electronics12061332.

[33] R. Bondaruc, N. Schnepf, R. Badonnel, C. A. Ardagna, and M. Anisetti. "Vulnerability-Aware Secure Service Deployment in Cloud-Edge Continuum".
In: *IEEE Transactions on Network and Service Management* (2025), pp. 1–1. DOI: 10.1109/TNSM.2025.3606624.

[34] A. Botta, W. de Donato, V. Persico, and A. Pescapé. "Integration of Cloud Computing and Internet of Things: A Survey".
In: *Future Generation Computer Systems* 56 (Mar. 2016), pp. 684–700. DOI: 10.1016/j.future.2015.09.021.

[35] A. Brogi and S. Forti. "QoS-Aware Deployment of IoT Applications Through the Fog".
In: *IEEE Internet of Things Journal* 4.5 (Oct. 2017), pp. 1185–1192.

[36] A. Brogi, S. Forti, C. Guerrero, and I. Lera. "Meet Genetic Algorithms in Monte Carlo: Optimised Placement of Multi-Service Applications in the Fog".
In: *2019 IEEE International Conference on Edge Computing (EDGE)*. July 2019, pp. 13–17. DOI: 10.1109/EDGE.2019.00016.

[37] R. A. C. da Silva and N. L. S. da Fonseca. "On the Location of Fog Nodes in Fog-Cloud Infrastructures".
In: *Sensors* 19.11 (Jan. 2019), p. 2445. DOI: 10.3390/s19112445.

[38] J. Caballero, Z. Liang, P. Poosankam, and D. Song. "Towards Generating High Coverage Vulnerability-Based Signatures with Protocol-Level Constraint-Guided Exploration". In: *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID)*. Saint-Malo, France: Springer-Verlag, 2009, pp. 161–181. DOI: 10.1007/978-3-642-04342-0_9.

[39] K. Cao, Y. Liu, G. Meng, and Q. Sun. "An Overview on Edge Computing Research".
In: *IEEE Access* 8 (2020), pp. 85714–85728. DOI: 10.1109/ACCESS.2020.2991734.

[40] V. Casola, A. D. Benedictis, S. D. Martino, N. Mazzocca, and L. L. L. Starace. "Security-Aware Deployment Optimization of Cloud–Edge Systems in Industrial IoT".
In: *IEEE Internet of Things Journal* 8.16 (Aug. 2021), pp. 12724–12733.

[41] G. Castellano, F. Esposito, and F. Risso. “A Distributed Orchestration Algorithm for Edge Computing Resources with Guarantees”. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. Apr. 2019, pp. 2548–2556. DOI: 10.1109/INFOCOM.2019.8737532.

[42] B. Charyyev, E. Arslan, and M. H. Gunes. “Latency Comparison of Cloud Datacenters and Edge Servers”. In: *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*. Dec. 2020, pp. 1–6. DOI: 10.1109/GLOBECOM42002.2020.9322406.

[43] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, and Q. Zhang. “Edge Computing in IoT-Based Manufacturing”. In: *IEEE Communications Magazine* 56.9 (Sept. 2018), pp. 103–109. DOI: 10.1109/MCOM.2018.1701231.

[44] L. Chen, Y. Bai, P. Zhou, Y. Li, Z. Qu, and J. Xu. “On Adaptive Edge Microservice Placement: A Reinforcement Learning Approach Endowed With Graph Comprehension”. In: *IEEE Transactions on Mobile Computing* 23.12 (Dec. 2024), pp. 11144–11158. DOI: 10.1109/TMC.2024.3396510.

[45] M.-H. Chen, M. Dong, and B. Liang. “Resource Sharing of a Computing Access Point for Multi-User Mobile Cloud Offloading with Delay Constraints”. In: *IEEE Transactions on Mobile Computing* 17.12 (Dec. 2018), pp. 2868–2881. DOI: 10.1109/TMC.2018.2815533.

[46] M.-H. Chen, B. Liang, and M. Dong. “Multi-User Multi-Task Offloading and Resource Allocation in Mobile Cloud Systems”. In: *IEEE Transactions on Wireless Communications* 17.10 (Oct. 2018), pp. 6790–6805. DOI: 10.1109/TWC.2018.2864559.

[47] X. Chen, W. Li, S. Lu, Z. Zhou, and X. Fu. “Efficient Resource Allocation for On-Demand Mobile-Edge Cloud Computing”. In: *IEEE Transactions on Vehicular Technology* 67.9 (Sept. 2018), pp. 8769–8780. DOI: 10.1109/TVT.2018.2846232.

[48] Y. Chen, Y. Sun, B. Yang, and T. Taleb. “Joint Caching and Computing Service Placement for Edge-Enabled IoT Based on Deep Reinforcement Learning”. In: *IEEE Internet of Things Journal* 9.19 (Oct. 2022), pp. 19501–19514. DOI: 10.1109/JIOT.2022.3168869.

[49] Y. Chen, J. Zhao, Y. Wu, J. Huang, and X. Shen.
"QoE-Aware Decentralized Task Offloading and Resource Allocation for End-Edge-Cloud Systems: A Game-Theoretical Approach".
In: *IEEE Transactions on Mobile Computing* 23.1 (Jan. 2024), pp. 769–784.
DOI: 10.1109/TMC.2022.3223119.

[50] J.-S. Choi, Y.-Y. Cho, and J.-Y. Choi. "The Design of a Context-Aware Workflow Language for Supporting Multiple Workflows".
In: *Journal of Internet Computing and Services* 10.6 (2009), pp. 145–157.

[51] M. R. Crusoe et al. "Methods included: standardizing computational reuse and portability with the Common Workflow Language".
In: *Commun. ACM* 65.6 (May 2022), pp. 54–63. ISSN: 0001-0782.
DOI: 10.1145/3486897. URL: https://doi.org/10.1145/3486897.

[52] A. Das, A. Leaf, C. A. Varela, and S. Patterson. "Skedulix: Hybrid Cloud Scheduling for Cost-Efficient Execution of Serverless Applications".
In: *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*.
ISSN: 2159-6190. Oct. 2020, pp. 609–618.

[53] R. David et al. "TensorFlow Lite Micro: Embedded Machine Learning for TinyML Systems". In: *Proceedings of Machine Learning and Systems*.
Ed. by A. Smola, A. Dimakis, and I. Stoica. Vol. 3. 2021, pp. 800–811.

[54] R. Deng, R. Lu, C. Lai, and T. H. Luan.
"Towards Power Consumption-Delay Tradeoff by Workload Allocation in Cloud-Fog Computing".
In: *2015 IEEE International Conference on Communications (ICC)*. June 2015, pp. 3909–3914. DOI: 10.1109/ICC.2015.7248934.

[55] S. Dhar, J. Guo, J. (Liu, S. Tripathi, U. Kurup, and M. Shah.
"A Survey of On-Device Machine Learning: An Algorithms and Learning Theory Perspective".
In: *ACM Trans. Internet Things* 2.3 (July 2021), 15:1–15:49.
DOI: 10.1145/3450494.

[56] M. Dias de Assunção, A. da Silva Veith, and R. Buyya.
"Distributed Data Stream Processing and Edge Computing: A Survey on Resource Elasticity and Future Directions".
In: *Journal of Network and Computer Applications* 103 (Feb. 2018), pp. 1–17.
DOI: 10.1016/j.jnca.2017.12.001.

[57] W. Ding, F. Luo, L. Han, C. Gu, H. Lu, and J. Fuentes.
"Adaptive Virtual Machine Consolidation Framework Based on Performance-to-Power Ratio in Cloud Data Centers".
In: *Future Generation Computer Systems* 111 (Oct. 2020), pp. 254–270.
DOI: 10.1016/j.future.2020.05.004.

[58] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek. "Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling".
In: *IEEE Transactions on Communications* 65.8 (Aug. 2017), pp. 3571–3584.
DOI: 10.1109/TCOMM.2017.2699660.

[59] S. Disabato and M. Roveri.
"Incremental On-Device Tiny Machine Learning".
In: *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things.*
AIChallengeIoT '20.
New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 7–13. DOI: 10.1145/3417313.3429378.

[60] W. Du, T. Lei, Q. He, W. Liu, Q. Lei, H. Zhao, and W. Wang.
Service Capacity Enhanced Task Offloading and Resource Allocation in Multi-Server Edge Computing Environment. Mar. 2019.
DOI: 10.48550/arXiv.1903.04709. arXiv: 1903.04709 [cs].

[61] T. L. Duc, R. G. Leiva, P. Casari, and P.-O. Östberg.
"Machine Learning Methods for Reliable Resource Provisioning in Edge-Cloud Computing: A Survey".
In: *ACM Comput. Surv.* 52.5 (Sept. 2019), 94:1–94:39.
DOI: 10.1145/3341145.

[62] A. Al-Dulaimy et al. "The Computing Continuum: From IoT to the Cloud".
In: *Internet of Things* 27 (Oct. 2024), p. 101272.
DOI: 10.1016/j.iot.2024.101272.

[63] M. S. Elbamby, M. Bennis, and W. Saad.
"Proactive Edge Computing in Latency-Constrained Fog Networks".
In: *2017 European Conference on Networks and Communications (EuCNC).*
June 2017, pp. 1–6. DOI: 10.1109/EuCNC.2017.7980678.

[64] M. S. Elbamby, M. Bennis, W. Saad, M. Latva-aho, and C. S. Hong.
"Proactive Edge Computing in Fog Networks with Latency and Reliability Guarantees". In: *EURASIP Journal on Wireless Communications and Networking* 2018.1 (Aug. 2018), p. 209. DOI: 10.1186/s13638-018-1218-y.

[65] ETSI. *GS MEC 003 Multi-access Edge Computing (MEC); Framework and Reference Architecture*. V3.1.1, Mar. 2022.

[66] W. Fan. "Blockchain-Secured Task Offloading and Resource Allocation for Cloud-Edge-End Cooperative Networks".
In: *IEEE Transactions on Mobile Computing* 23.8 (Aug. 2024), pp. 8092–8110.
DOI: 10.1109/TMC.2023.3342817.

[67] V. Farhadi, F. Mehmeti, T. He, T. F. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis. "Service Placement and Request Scheduling for Data-Intensive Applications in Edge Clouds".
In: *IEEE/ACM Transactions on Networking* 29.2 (Apr. 2021), pp. 779–792.
ISSN: 1558-2566. DOI: 10.1109/TNET.2020.3048613.

[68] I. Farris, L. Militano, M. Nitti, L. Atzori, and A. Iera. "MIFaaS: A Mobile-IoT-Federation-as-a-Service Model for Dynamic Cooperation of IoT Cloud Providers".
In: *Future Generation Computer Systems* 70 (May 2017), pp. 126–137.
DOI: 10.1016/j.future.2016.06.028.

[69] F. Faticanti, F. De Pellegrini, D. Siracusa, D. Santoro, and S. Cretti. "Cutting Throughput with the Edge: App-Aware Placement in Fog Computing".
In: *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. June 2019, pp. 196–203.
DOI: 10.1109/CSCloud/EdgeCom.2019.00026.

[70] N. Fernando, S. W. Loke, and W. Rahayu. "Mobile Cloud Computing: A Survey".
In: *Future Generation Computer Systems*. Including Special Section: AIRCC-NetCoM 2009 and Special Section: Clouds and Service-Oriented Architectures 29.1 (Jan. 2013), pp. 84–106.
DOI: 10.1016/j.future.2012.05.023.

[71] C. P. Filho, E. Marques, V. Chang, L. dos Santos, F. Bernardini, P. F. Pires, L. Ochi, and F. C. Delicato. "A Systematic Literature Review on Distributed Machine Learning in Edge Computing".
In: *Sensors* 22.7 (Jan. 2022), p. 2665. DOI: 10.3390/s22072665.

[72] P. Foreman. *Vulnerability Management*. Taylor & Francis Group, 2010.

[73] S. Forti, G.-L. Ferrari, and A. Brogi. "Secure Cloud-Edge Deployments, with Trust".
In: *Future Generation Computer Systems* 102 (Jan. 2020), pp. 775–788.
ISSN: 0167-739X. DOI: 10.1016/j.future.2019.08.020.

[74] K. Fu, W. Zhang, Q. Chen, D. Zeng, and M. Guo. “Adaptive Resource Efficient Microservice Deployment in Cloud-Edge Continuum”. In: *IEEE Transactions on Parallel and Distributed Systems* 33.8 (Aug. 2022), pp. 1825–1840.

[75] N. Gholipour, E. Arianyan, and R. Buyya. “A Novel Energy-Aware Resource Management Technique Using Joint VM and Container Consolidation Approach for Green Computing in Cloud Data Centers”. In: *Simulation Modelling Practice and Theory* 104 (Nov. 2020), p. 102127. DOI: 10.1016/j.simpat.2020.102127.

[76] S. Giallorenzo, F. Montesi, and M. Peressotti. “Choral: Object-oriented Choreographic Programming”. In: *ACM Trans. Program. Lang. Syst.* 46.1 (Jan. 2024), 1:1–1:59. DOI: 10.1145/3632398.

[77] F. Giannone, P. A. Frangoudis, A. Ksentini, and L. Valcarenghi. “Orchestrating heterogeneous MEC-based applications for connected vehicles”. In: *Computer Networks* 180 (Oct. 2020), p. 107402.

[78] A. E. Giannopoulos, I. Paralikas, S. T. Spantideas, and P. Trakadas. “HOODIE: Hybrid Computation Offloading via Distributed Deep Reinforcement Learning in Delay-Aware Cloud-Edge Continuum”. In: *IEEE Open Journal of the Communications Society* 5 (2024), pp. 7818–7841. DOI: 10.1109/OJCOMS.2024.3514456.

[79] P. Gkonis, A. Giannopoulos, P. Trakadas, X. Masip-Bruin, and F. D’Andria. “A Survey on IoT-Edge-Cloud Continuum Systems: Status, Challenges, Use Cases, and Open Issues”. In: *Future Internet* 15.12 (Dec. 2023), p. 383. DOI: 10.3390/fi15120383.

[80] M. Goudarzi, M. Palaniswami, and R. Buyya. “A Distributed Deep Reinforcement Learning Technique for Application Placement in Edge and Fog Computing Environments”. In: *IEEE Transactions on Mobile Computing* 22.5 (May 2023), pp. 2491–2505. DOI: 10.1109/TMC.2021.3123165.

[81] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya. “An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments”. In: *IEEE Transactions on Mobile Computing* 20.4 (Apr. 2021), pp. 1298–1311. ISSN: 1558-0660. DOI: 10.1109/TMC.2020.2967041.

[82] D. Gupta and J. Kuri.
"Optimal VNF Placement and Traffic Steering Using Column Generation".
In: *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*.
Nov. 2020, pp. 1–4. DOI: 10.1109/CloudNet51028.2020.9335707.

[83] E. E. Haber, H. A. Alameddine, C. Assi, and S. Sharafeddine.
"A Reliability-aware Computation Offloading Solution via UAV-mounted Cloudlets".
In: *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*.
Nov. 2019, pp. 1–6. DOI: 10.1109/CloudNet47604.2019.9064038.

[84] M. A. Hassan, M. Xiao, Q. Wei, and S. Chen.
"Help Your Mobile Applications with Fog Computing".
In: *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops)*.
June 2015, pp. 1–6. DOI: 10.1109/SECONW.2015.7328146.

[85] T. He and R. Buyya.
"A Taxonomy of Live Migration Management in Cloud Computing".
In: *ACM Computing Surveys* 56.3 (Oct. 2023). ISSN: 0360-0300.
DOI: 10.1145/3615353.

[86] X. He, J. Liu, R. Jin, and H. Dai.
"Privacy-Aware Offloading in Mobile-Edge Computing".
In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*.
Dec. 2017, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8253985.

[87] A. Hedhli and H. Mezni.
"A Survey of Service Placement in Cloud Environments". en.
In: *J Grid Computing* 19.3 (Sept. 2021), p. 23. ISSN: 1570-7873, 1572-9184.
DOI: 10.1007/s10723-021-09565-z.

[88] C.-H. Hong and B. Varghese. "Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms".
In: *ACM Comput. Surv.* 52.5 (Sept. 2019), 97:1–97:37.
DOI: 10.1145/3326066.

[89] A. Islam, A. Debnath, M. Ghose, and S. Chakraborty.
"A Survey on Task Offloading in Multi-access Edge Computing".
In: *Journal of Systems Architecture* 118 (Sept. 2021), p. 102225.
DOI: 10.1016/j.sysarc.2021.102225.

[90] Z. Jiang and S. Mao. "Energy Delay Tradeoff in Cloud Offloading for Multi-Core Mobile Devices". In: *IEEE Access* 3 (2015), pp. 2306–2316.
DOI: 10.1109/ACCESS.2015.2499300.

[91] H. Jin, X. Zhu, and C. Zhao.
"Computation Offloading Optimization Based on Probabilistic SFC for Mobile Online Gaming in Heterogeneous Network".
In: *IEEE Access* 7 (2019), pp. 52168–52180.
DOI: 10.1109/ACCESS.2019.2909971.

[92] S. Jošilo and G. Dán. "Wireless and Computing Resource Allocation for Selfish Computation Offloading in Edge Computing".
In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*.
Apr. 2019, pp. 2467–2475. DOI: 10.1109/INFOCOM.2019.8737480.

[93] A. Jula, E. Sundararajan, and Z. Othman.
"Cloud Computing Service Composition: A Systematic Literature Review".
In: *Expert Systems with Applications* 41.8 (June 2014), pp. 3809–3824.
DOI: 10.1016/j.eswa.2013.12.017.

[94] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, et al.
"MEC in 5G networks". In: *ETSI white paper* 28 (2018 2018), pp. 1–28.

[95] P. Keshavarz Haddadha, M. H. Rezvani, M. MollaMotalebi, and A. Shankar.
"Machine Learning Methods for Service Placement: A Systematic Review".
In: *Artificial Intelligence Review* 57.3 (Feb. 2024), p. 61.
DOI: 10.1007/s10462-023-10684-0.

[96] N. Kiran, X. Liu, S. Wang, and C. Yin. "VNF Placement and Resource Allocation in SDN/NFV-Enabled MEC Networks". In: *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*.
Apr. 2020, pp. 1–6. DOI: 10.1109/WCNCW48565.2020.9124910.

[97] L. Kong, J. Tan, J. Huang, G. Chen, S. Wang, X. Jin, P. Zeng, M. Khan, and S. K. Das. "Edge-Computing-Driven Internet of Things: A Survey".
In: *ACM Comput. Surv.* 55.8 (Dec. 2022), 174:1–174:41.
DOI: 10.1145/3555308.

[98] H. Kourani and S. J. van Zelst.
"POWL: Partially Ordered Workflow Language".
In: *Business Process Management*.
Ed. by C. Di Francescomarino, A. Burattin, C. Janiesch, and S. Sadiq.
Cham: Springer Nature Switzerland, 2023, pp. 92–108.
DOI: 10.1007/978-3-031-41620-0_6.

[99] D. Kreković, P. Krivić, I. Podnar Žarko, M. Kušek, and D. Le-Phuoc. "Reducing Communication Overhead in the IoT–Edge–Cloud Continuum: A Survey on Protocols and Data Reduction Strategies". In: *Internet of Things* 31 (May 2025), p. 101553. DOI: 10.1016/j.iot.2025.101553.

[100] A. Laghrissi, T. Taleb, M. Bagaa, and H. Flinck. "Towards Edge Slicing: VNF Placement Algorithms for a Dynamic & Realistic Edge Cloud Environment". In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. Dec. 2017, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254653.

[101] A. Langley et al. "The QUIC Transport Protocol: Design and Internet-Scale Deployment". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, Aug. 2017, pp. 183–196. DOI: 10.1145/3098822.3098842.

[102] K. Liolis, J. Cahill, E. Higgins, M. Corici, E. Troudt, and P. Sutton. "Over-the-air demonstration of satellite integration with 5G core network and multi-access edge computing use case". In: *IEEE 5G World Forum, 5GWF 2019 - Conference Proceedings*. Sept. 2019, pp. 1–5.

[103] H. Liu, S. Ding, S. Wang, G. Zhao, and C. Wang. "Multi-Objective Optimization Service Function Chain Placement Algorithm Based on Reinforcement Learning". In: *Journal of Network and Systems Management* 30.4 (Oct. 2022), p. 58. DOI: 10.1007/s10922-022-09673-5.

[104] T. Liu, S. Ni, X. Li, Y. Zhu, L. Kong, and Y. Yang. "Deep Reinforcement Learning Based Approach for Online Service Placement and Computation Resource Allocation in Edge Computing". In: *IEEE Transactions on Mobile Computing* 22.7 (July 2023), pp. 3870–3881. DOI: 10.1109/TMC.2022.3148254.

[105] Y. Liu and Q. Ma. "Service Placement Considering Robustness and Dynamic in Edge Computing". In: *2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. Apr. 2021, pp. 369–374. DOI: 10.1109/ICCCBDA51879.2021.9442568.

[106] Y. Liu, G. Feng, G. Zhao, Z. Chen, and S. Qin. "Virtual Network Function Deployment Strategy in Clustered Multi-Mobile Edge Clouds". In: *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. May 2020, pp. 1–6. DOI: 10.1109/WCNC45663.2020.9120702.

[107] F. Luo, S. Zheng, W. Ding, J. Fuentes, and Y. Li. "An Edge Server Placement Method Based on Reinforcement Learning". In: *Entropy* 24.3 (Mar. 2022), p. 317. DOI: 10.3390/e24030317.

[108] X. Lyu, H. Tian, C. Sengul, and P. Zhang. "Multiuser Joint Task Offloading and Resource Optimization in Proximate Clouds". In: *IEEE Transactions on Vehicular Technology* 66.4 (Apr. 2017), pp. 3435–3447. DOI: 10.1109/TVT.2016.2593486.

[109] H. Ma, Z. Zhou, and X. Chen. "Leveraging the Power of Prediction: Predictive Service Placement for Latency-Sensitive Mobile Edge Computing". In: *IEEE Transactions on Wireless Communications* 19.10 (Oct. 2020), pp. 6454–6468. DOI: 10.1109/TWC.2020.3003459.

[110] H. Ma, Z. Zhou, and X. Chen. "Predictive Service Placement in Mobile Edge Computing". In: *2019 IEEE/CIC International Conference on Communications in China (ICCC)*. Aug. 2019, pp. 792–797. DOI: 10.1109/ICCChina.2019.8855957.

[111] W. Ma, X. Liu, and L. Mashayekhy. "A Strategic Game for Task Offloading among Capacitated UAV-Mounted Cloudlets". In: *2019 IEEE International Congress on Internet of Things (ICIOT)*. July 2019, pp. 61–68. DOI: 10.1109/ICIOT.2019.00022.

[112] A. Mehta and E. Elmroth. "Distributed Cost-Optimized Placement for Latency-Critical Applications in Heterogeneous Environments". In: *2018 IEEE International Conference on Autonomic Computing (ICAC)*. Sept. 2018, pp. 121–130. DOI: 10.1109/ICAC.2018.00022.

[113] *MITRE Corporation*. https://www.mitre.org.

[114] MITRE Corporation. *CVE Common Vulnerability Exposure*. https://www.cve.org/.

[115] MITRE Corporation. *OVAL: Open Vulnerabilities Assessment Language*. https://oval.mitre.org.

[116] A. Mohamad and H. S. Hassanein. "On Demonstrating the Gain of SFC Placement with VNF Sharing at the Edge". In: *2019 IEEE Global Communications Conference (GLOBECOM)*. Dec. 2019, pp. 1–6. DOI: 10.1109/GLOBECOM38437.2019.9014106.

[117] S. Moreschini, F. Pecorelli, X. Li, S. Naz, D. Hästbacka, and D. Taibi. "Cloud Continuum: The Definition".
In: *IEEE Access* 10 (2022), pp. 131876–131886.
DOI: 10.1109/ACCESS.2022.3229185.

[118] J. Moura and D. Hutchison. "Game Theory for Multi-Access Edge Computing: Survey, Use Cases, and Future Trends".
In: *IEEE Communications Surveys & Tutorials* 21.1 (2019), pp. 260–288.
DOI: 10.1109/COMST.2018.2863030.

[119] C. Mouradian, S. Kianpisheh, M. Abu-Lebdeh, F. Ebrahimnezhad, N. T. Jahromi, and R. H. Glitho. "Application Component Placement in NFV-Based Hybrid Cloud/Fog Systems With Mobile Fog Nodes".
In: *IEEE Journal on Selected Areas in Communications* 37.5 (May 2019), pp. 1130–1143. DOI: 10.1109/JSAC.2019.2906790.

[120] *MQTT - The Standard for IoT Messaging*. https://mqtt.org/.

[121] G. Myers. *The Art of Software Testing*. Wiley Publishing, 2015.

[122] S. Nastic, P. Raith, A. Furutanpey, T. Pusztai, and S. Dustdar. "A Serverless Computing Fabric for Edge & Cloud". In: *2022 IEEE 4th International Conference on Cognitive Machine Intelligence (CogMI)*. 2022, pp. 1–12.

[123] B. Németh, N. Molner, J. Martín-Pérez, C. J. Bernardos, A. de la Oliva, and B. Sonkoly. "Delay and Reliability-Constrained VNF Placement on Mobile and Volatile 5G Infrastructure".
In: *IEEE Transactions on Mobile Computing* 21.9 (Sept. 2022), pp. 3150–3162.
DOI: 10.1109/TMC.2021.3055426.

[124] B. Németh, Y.-A. Pignolet, M. Rost, S. Schmid, and B. Vass. "Cost-Efficient Embedding of Virtual Networks With and Without Routing Flexibility".
In: *2020 IFIP Networking Conference (Networking)*. June 2020, pp. 476–484.

[125] B. Németh, B. Sonkoly, M. Rost, and S. Schmid. "Efficient Service Graph Embedding: A Practical Approach".
In: *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. Nov. 2016, pp. 19–25.
DOI: 10.1109/NFV-SDN.2016.7919470.

[126] P.-D. Nguyen and L. B. Le. "Joint Computation Offloading, SFC Placement, and Resource Allocation for Multi-Site MEC Systems". In: *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. May 2020, pp. 1–6. DOI: 10.1109/WCNC45663.2020.9120597.

[127] J. Ni, K. Zhang, and A. V. Vasilakos.
"Security and Privacy for Mobile Edge Caching: Challenges and Solutions".
In: *IEEE Wireless Communications* 28.3 (June 2021), pp. 77–83.
DOI: 10.1109/MWC.001.2000329.

[128] M. S. Nikoo, Ö. Babur, and M. Van Den Brand.
"A Survey on Service Composition Languages".
In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*.
Virtual Event Canada: ACM, Oct. 2020, pp. 1–5.
DOI: 10.1145/3417990.3421402.

[129] T. H. Noor, S. Zeadally, A. Alfazi, and Q. Z. Sheng.
"Mobile Cloud Computing: Challenges and Future Research Directions".
In: *Journal of Network and Computer Applications* 115 (Aug. 2018),
pp. 70–85. DOI: 10.1016/j.jnca.2018.04.018.

[130] A. F. Ocampo and J. Santos. "Reinforcement Learning-Driven Service Placement in 6G Networks across the Compute Continuum". In: *2024 20th International Conference on Network and Service Management (CNSM)*.
Oct. 2024, pp. 1–9. DOI: 10.23919/CNSM62983.2024.10814365.

[131] A. Orive, A. Agirre, H.-L. Truong, I. Sarachaga, and M. Marcos. "Quality of Service Aware Orchestration for Cloud–Edge Continuum Applications".
In: *Sensors* 22.5 (Jan. 2022), p. 1755.

[132] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang.
"Adaptive User-managed Service Placement for Mobile Edge Computing: An Online Learning Approach".
In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*.
Apr. 2019, pp. 1468–1476. DOI: 10.1109/INFOCOM.2019.8737560.

[133] J. Pan and J. McElhannon. "Future Edge Cloud and Edge Computing for Internet of Things Applications".
In: *IEEE Internet of Things Journal* 5.1 (Feb. 2018), pp. 439–449.
DOI: 10.1109/JIOT.2017.2767608.

[134] S. Pasteris, S. Wang, M. Herbster, and T. He. "Service Placement with Provable Guarantees in Heterogeneous Edge Computing Systems".
In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*.
Apr. 2019, pp. 514–522. DOI: 10.1109/INFOCOM.2019.8737449.

[135] E. Peltonen, A. Sojan, and T. Päivärinta. "Towards Real-time Learning for Edge-Cloud Continuum with Vehicular Computing". In: *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*. 2021, pp. 921–926. DOI: 10.1109/WF-IoT51360.2021.9595628.

[136] Q.-V. Pham, L. B. Le, S.-H. Chung, and W.-J. Hwang. "Mobile Edge Computing With Wireless Backhaul: Joint Task Offloading and Resource Allocation". In: *IEEE Access* 7 (2019), pp. 16444–16459. DOI: 10.1109/ACCESS.2018.2883692.

[137] Q.-V. Pham, T. Leanh, N. H. Tran, B. J. Park, and C. S. Hong. "Decentralized Computation Offloading and Resource Allocation for Mobile-Edge Computing: A Matching Game Approach". In: *IEEE Access* 6 (2018), pp. 75868–75885. DOI: 10.1109/ACCESS.2018.2882800.

[138] J. Plachy, Z. Becvar, and E. C. Strinati. "Dynamic Resource Allocation Exploiting Mobility Prediction in Mobile Edge Computing". In: *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. Sept. 2016, pp. 1–6. DOI: 10.1109/PIMRC.2016.7794955.

[139] F. Poltronieri, C. Stefanelli, M. Tortonesi, and M. Zaccarini. "Reinforcement Learning vs. Computational Intelligence: Comparing Service Management Approaches for the Cloud Continuum". In: *Future Internet* 15.11 (Nov. 2023), p. 359. DOI: 10.3390/fi15110359.

[140] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb. "Survey on Multi-Access Edge Computing for Internet of Things Realization". In: *IEEE Communications Surveys & Tutorials* 20.4 (2018), pp. 2961–2991. DOI: 10.1109/COMST.2018.2849509.

[141] C. Puliafito, L. Conforti, A. Virdis, and E. Mingozzi. "Server-Side QUIC Connection Migration to Support Microservice Deployment at the Edge". In: *Pervasive and Mobile Computing* 83 (July 2022), p. 101580. DOI: 10.1016/j.pmcj.2022.101580.

[142] G. Qu, H. Wu, R. Li, and P. Jiao. "DMRO: A Deep Meta Reinforcement Learning-Based Task Offloading Framework for Edge-Cloud Computing". In: *IEEE Transactions on Network and Service Management* 18.3 (Sept. 2021), pp. 3448–3459. DOI: 10.1109/TNSM.2021.3087258.

[143] J. G. Quenum and J. Josua.
"Multi-Cloud Serverless Function Composition". In: *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing*.
UCC '21.
Leicester, United Kingdom: Association for Computing Machinery, 2021, pp. 1–10.

[144] S. Rac and M. Brorsson. "Cost-Aware Service Placement and Scheduling in the Edge-Cloud Continuum". In: *ACM Transactions on Architecture and Code Optimization* 21.2 (June 2024), pp. 1–24. DOI: 10.1145/3640823.

[145] P. Raith, S. Nastic, and S. Dustdar.
"Serverless Edge Computing—Where We Are and What Lies Ahead".
In: *IEEE Internet Computing* 27.3 (May 2023), pp. 50–64.
DOI: 10.1109/MIC.2023.3260939.

[146] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato.
"Machine Learning Meets Computation and Communication Control in Evolving Edge and Cloud: Challenges and Future Perspective".
In: *IEEE Communications Surveys & Tutorials* 22.1 (2020), pp. 38–67.
DOI: 10.1109/COMST.2019.2943405.

[147] M. Rost and S. Schmid.
"Charting the Complexity Landscape of Virtual Network Embeddings".
In: *2018 IFIP Networking Conference (IFIP Networking) and Workshops*.
May 2018, pp. 1–9. DOI: 10.23919/IFIPNetworking.2018.8696604.

[148] G. R. Russo, V. Cardellini, and F. L. Presti. "Serverless Functions in the Cloud-Edge Continuum: Challenges and Opportunities".
In: *2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. Mar. 2023, pp. 321–328.
DOI: 10.1109/PDP59025.2023.00056.

[149] D. Saha.
"Extending Logical Attack Graphs for Efficient Vulnerability Analysis".
In: *Proceedings of the 15th ACM Conference on Computer and Communications Security (ACM CCS)*.
Alexandria, Virginia, USA: ACM, 2008, pp. 63–74.

[150] F. A. Salaht, F. Desprez, and A. Lebre. "An Overview of Service Placement Problem in Fog and Edge Computing". en.
In: *ACM Comput. Surv.* 53.3 (May 2021), pp. 1–35.
ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3391196.

[151] H. Sami, A. Mourad, H. Otrok, and J. Bentahar. "Demand-Driven Deep Reinforcement Learning for Scalable Fog and Service Placement". In: *IEEE Transactions on Services Computing* 15.5 (Sept. 2022), pp. 2671–2684. DOI: 10.1109/TSC.2021.3075988.

[152] F. Samie, V. Tsoutsouras, L. Bauer, S. Xydis, D. Soudris, and J. Henkel. "Computation Offloading and Resource Allocation for Low-Power IoT Edge Devices". In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. Dec. 2016, pp. 7–12. DOI: 10.1109/WF-IoT.2016.7845499.

[153] S. Sardellitti, G. Scutari, and S. Barbarossa. "Joint Optimization of Radio and Computational Resources for Multicell Mobile-Edge Computing". In: *IEEE Transactions on Signal and Information Processing over Networks* 1.2 (June 2015), pp. 89–103. DOI: 10.1109/TSIPN.2015.2448520.

[154] J. Sauve, R. Santos, R. Reboucas, and A. Moura. "Change Priority Determination in IT Service Management Based on Risk Exposure". In: *IEEE Transactions on Network and Service Management* 5.3 (Sept. 2008), pp. 178–187. ISSN: 1932-4537. DOI: 10.1109/TNSM.2009.031105.

[155] A. Singh, P. Vepakomma, O. Gupta, and R. Raskar. *Detailed Comparison of Communication Efficiency of Split Learning and Federated Learning*. Sept. 2019. DOI: 10.48550/arXiv.1909.09145. arXiv: 1909.09145 [cs].

[156] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner. "Optimized IoT service placement in the fog". en. In: *SOCA* 11.4 (Dec. 2017), pp. 427–443. ISSN: 1863-2394. DOI: 10.1007/s11761-017-0219-8.

[157] Y. Song, S. S. Yau, R. Yu, X. Zhang, and G. Xue. "An Approach to QoS-based Task Distribution in Edge Computing Networks for IoT Applications". In: *2017 IEEE International Conference on Edge Computing (EDGE)*. June 2017, pp. 32–39. DOI: 10.1109/IEEE.EDGE.2017.50.

[158] B. Sonkoly, J. Czentye, M. Szalay, B. Németh, and L. Toka. "Survey on Placement Methods in the Edge and Beyond". In: *IEEE Communications Surveys & Tutorials* 23.4 (2021), pp. 2590–2629. ISSN: 1553-877X. DOI: 10.1109/COMST.2021.3101460.

[159] B. Sonkoly, R. Szabó, B. Németh, J. Czentye, D. Haja, M. Szalay, J. Dóka, B. P. Gerő, D. Jocha, and L. Toka.
"5G Applications From Vision to Reality: Multi-Operator Orchestration".
In: *IEEE Journal on Selected Areas in Communications* 38.7 (July 2020), pp. 1401–1416. DOI: 10.1109/JSAC.2020.2999684.

[160] P. Soumplis, P. Kokkinos, A. Kretsis, P. Nicopolitidis, G. Papadimitriou, and E. Varvarigos.
"Resource Allocation Challenges in the Cloud and Edge Continuum".
In: *Advances in Computing, Informatics, Networking and Cybersecurity: A Book Honoring Professor Mohammad S. Obaidat's Significant Scientific Contributions*.
Ed. by P. Nicopolitidis, S. Misra, L. T. Yang, B. Zeigler, and Z. Ning.
Cham: Springer International Publishing, 2022, pp. 443–464.
DOI: 10.1007/978-3-030-87049-2_15.

[161] H. Sun, H. Yu, G. Fan, L. Chen, and Z. Liu.
"Security-Aware and Time-Guaranteed Service Placement in Edge Clouds".
In: *IEEE Transactions on Network and Service Management* 20.1 (Mar. 2023), pp. 711–725. ISSN: 1932-4537. DOI: 10.1109/TNSM.2022.3213761.

[162] M. Szabolcs, D. Haja, and M. Szalay. "Cost-Efficient Resource Allocation Method for Heterogeneous Cloud Environments".
In: *INFOCOMMUNICATIONS JOURNAL* 10.1 (2018), pp. 15–21.
DOI: 10/1/InfocomJ_2018_1_3_Szabo.pdf.

[163] T. Taleb, M. Bagaa, and A. Ksentini. "User Mobility-Aware Virtual Network Function Placement for Virtual 5G Network Infrastructure".
In: *2015 IEEE International Conference on Communications (ICC)*. June 2015, pp. 3879–3884. DOI: 10.1109/ICC.2015.7248929.

[164] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella.
"On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration".
In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1657–1681.
DOI: 10.1109/COMST.2017.2705720.

[165] H. Tang, Y. Wang, and X. Chen. "Network-Flow Algorithms for Virtual Service Placements in Mobile Edge Networks".
In: *2021 IEEE 6th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*. Apr. 2021, pp. 364–368.
DOI: 10.1109/ICCCBDA51879.2021.9442549.

[166] L. Toka, D. Haja, A. Kőrösi, and B. Sonkoly. "Resource Provisioning for Highly Reliable and Ultra-Responsive Edge Applications". In: *2019 IEEE 8th International Conference on Cloud Networking (CloudNet)*. Nov. 2019, pp. 1–6. DOI: 10.1109/CloudNet47604.2019.9064131.

[167] M. Trabelsi, N. M. Mehdi Bendaoud, and S. B. Ahmed. "Multi-objective Optimization for Dynamic Service Placement Strategy for Real-Time Applications in Fog Infrastructure". In: *2023 IEEE Symposium on Computers and Communications (ISCC)*. July 2023, pp. 607–612. DOI: 10.1109/ISCC58397.2023.10217950.

[168] T. X. Tran and D. Pompili. "Joint Task Offloading and Resource Allocation for Multi-Server Mobile-Edge Computing Networks". In: *IEEE Transactions on Vehicular Technology* 68.1 (Jan. 2019), pp. 856–868. DOI: 10.1109/TVT.2018.2881191.

[169] W. M. P. van der Aalst and A. H. M. ter Hofstede. "YAWL: Yet Another Workflow Language". In: *Information Systems* 30.4 (June 2005), pp. 245–275. DOI: 10.1016/j.is.2004.02.002.

[170] A. Vinel, J. Breu, T. H. Luan, and H. Hu. "Emerging Technology for 5G-enabled Vehicular Networks". In: *IEEE Wireless Communications* 24.6 (Dec. 2017), pp. 12–12. DOI: 10.1109/MWC.2017.8246820.

[171] S. Vinoski. "Advanced Message Queuing Protocol". In: *IEEE Internet Computing* 10.6 (Nov. 2006), pp. 87–89. DOI: 10.1109/MIC.2006.116.

[172] K. Voss, G. V. der Auwera, and J. Gentry. "Full-Stack Genomics Pipelining with GATK4 + WDL + Cromwell". In: *F1000Research* 6 (Aug. 2017). DOI: 10.7490/f1000research.1114634.1.

[173] T. T. Vu, D. N. Nguyen, D. T. Hoang, and E. Dutkiewicz. "QoS-Aware Fog Computing Resource Allocation Using Feasibility-Finding Benders Decomposition". In: *2019 IEEE Global Communications Conference (GLOBECOM)*. Dec. 2019, pp. 1–6. DOI: 10.1109/GLOBECOM38437.2019.9013527.

[174] M. Wang, B. Cheng, and J. Chen. "An Efficient Service Function Chaining Placement Algorithm in Mobile Edge Computing". In: *ACM Trans. Internet Technol.* 20.4 (Oct. 2020), 32:1–32:21. DOI: 10.1145/3388241.

[175] M. Wang, B. Cheng, W. Feng, and J. Chen. "An Efficient Service Function Chain Placement Algorithm in a MEC-NFV Environment". In: *2019 IEEE Global Communications Conference (GLOBECOM)*. Dec. 2019, pp. 1–6. DOI: 10.1109/GLOBECOM38437.2019.9013235.

[176] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung. "Dynamic Service Placement for Mobile Micro-Clouds with Predicted Future Costs". In: *IEEE Transactions on Parallel and Distributed Systems* 28.4 (Apr. 2017), pp. 1002–1016. DOI: 10.1109/TPDS.2016.2604814.

[177] S. Wang, M. Zafer, and K. K. Leung. "Online Placement of Multi-Component Applications in Edge Computing Environments". In: *IEEE Access* 5 (2017), pp. 2514–2533. DOI: 10.1109/ACCESS.2017.2665971.

[178] T. Wang, Q. Fan, X. Li, X. Zhang, Q. Xiong, S. Fu, and M. Gao. "DRL-SFCP: Adaptive Service Function Chains Placement with Deep Reinforcement Learning". In: *ICC 2021 - IEEE International Conference on Communications*. Montreal, QC, Canada: IEEE, June 2021, pp. 1–6. DOI: 10.1109/ICC42927.2021.9500964.

[179] T. Wang, X. Wei, T. Liang, and J. Fan. "Dynamic Tasks Scheduling Based on Weighted Bi-Graph in Mobile Cloud Computing". In: *Sustainable Computing: Informatics and Systems* 19 (Sept. 2018), pp. 214–222. DOI: 10.1016/j.suscom.2018.05.004.

[180] S. K. Woldeyes, Y. Zhang, W. Wang, and Z. Li. "Joint Resource Placement and Service Replica Placement Scheme in Mobile Edge Computing". In: *2023 IEEE ISPA/BDCloud/SocialCom/SustainCom*. Dec. 2023, pp. 360–367.

[181] R. Xie, Q. Tang, S. Qiao, H. Zhu, F. R. Yu, and T. Huang. "When Serverless Computing Meets Edge Computing: Architecture, Challenges, and Open Issues". In: *IEEE Wireless Communications* 28.5 (Oct. 2021), pp. 126–133. DOI: 10.1109/MWC.001.2000466.

[182] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang. "A Comprehensive Survey of Network Function Virtualization". In: *Computer Networks* 133 (Mar. 2018), pp. 212–262. DOI: 10.1016/j.comnet.2018.01.021.

[183] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang. "A Survey on the Edge Computing for the Internet of Things". In: *IEEE Access* 6 (2018), pp. 6900–6919. DOI: 10.1109/ACCESS.2017.2778504.

[184] H. Zhang, Y. Xiao, S. Bu, D. Niyato, R. Yu, and Z. Han. "Fog Computing in Multi-Tier Data Center Networks: A Hierarchical Game Approach". In: *2016 IEEE International Conference on Communications (ICC)*. May 2016, pp. 1–6. DOI: 10.1109/ICC.2016.7511146.

[185] J. Zhang, H. Huang, and X. Wang. "Resource Provision Algorithms in Cloud Computing: A Survey". In: *Journal of Network and Computer Applications* 64 (Apr. 2016), pp. 23–42. DOI: 10.1016/j.jnca.2015.12.018.

[186] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri. "Hetero-Edge: Orchestration of Real-time Vision Applications on Heterogeneous Edge Clouds". In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. Apr. 2019, pp. 1270–1278. DOI: 10.1109/INFOCOM.2019.8737478.

[187] H. Zhao, S. Deng, C. Zhang, W. Du, Q. He, and J. Yin. "A Mobility-Aware Cross-Edge Computation Offloading Framework for Partitionable Applications". In: *2019 IEEE International Conference on Web Services (ICWS)*. July 2019, pp. 193–200. DOI: 10.1109/ICWS.2019.00041.

[188] H. Zhu and C. Huang. "Availability-Aware Mobile Edge Application Placement in 5G Networks". In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. Dec. 2017, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254591.

[189] S. Zhu, L. Gui, J. Chen, Q. Zhang, and N. Zhang. "Cooperative Computation Offloading for UAVs: A Joint Radio and Computing Resource Allocation Approach". In: *2018 IEEE International Conference on Edge Computing (EDGE)*. July 2018, pp. 74–79. DOI: 10.1109/EDGE.2018.00017.