Using Knowledge Graphs to Automate Network Compliance of Containerized Services

Aleksandra Simić*, David Palma*
*Department of Information Security and Communication Technology
NTNU—Norwegian University of Science and Technology
Trondheim, Norway
{aleksandra.simic, david.palma}@ntnu.no

***Abstract*—Information and Communication Technology (ICT) infrastructures are increasingly adopting software-based approaches, using paradigms such as Infrastructure as Code (IaC). This is aligned with the design of modern systems, such as 5G with the use of Virtual Network Functions (VNFs) and containerized services, combining the expertise and participation of various stakeholders from diverse cultural and educational backgrounds, potentially across different organizations. To create a common understanding of containerized networking infrastructures, we developed an ontology that allows a unified representation of core networking concepts. By enriching pre-deployment tasks with semantics, we universally represented and verified different network policies based on formal logic. To validate our approach, we constructed two knowledge graphs using open-source 5G Core Network implementations and demonstrated the potential of our approach by checking for compliance with automated reasoning. Using our defined rules, we extracted knowledge about non-compliant services and made inferences based on the well-defined concepts in the ontology. Our findings suggest that different network security policies can be integrated into each knowledge base, contributing to autonomic and cognitive management of future infrastructures. This can potentially provide more reliable IaC definitions, improve machine interpretation of IaC deployments, and assist human actors in making better-informed decisions.**

***Index Terms*—Ontologies, Knowledge Management, Knowledge Representation, Formalisms and Methods, Infrastructure protection**

I. Introduction

In recent years, Information and Communication Technology (ICT) infrastructures have shifted from a legacy environment to a software-based approach, which improves flexibility and scalability [1]. By utilizing software-based infrastructure and automation, network engineers reduce the time needed for provisioning and configuring network elements using the Infrastructure as Code (IaC) principles [2]. Various actors can create configuration files that capture desired requirements, enabling version control and different Application Programming Interfaces (APIs) for infrastructure management. In addition, this approach leverages virtualized and containerized resources for simplified control, transparency and reusability.

Network programmability simplifies network management by minimizing the number of tasks usually performed manually. However, managing the knowledge of hybrid systems can be challenging since newly deployed network infrastructures coexist with legacy architectures. For instance, the fifth generation of cellular networks (5G) networks are currently deployed in parallel with older systems while adopting modern networking concepts such as Virtual Network Functions (VNFs) [3]. Moreover, network programmability integrated with deployment as code brings new software development practices, such as Continuous Integration and Deployment (CI/CD), into the networking domain. This shift introduces challenges in managing the dependencies between software-based networking components.

As networking systems are growing in size and capabilities, it is essential to comprehend abstract knowledge and make conclusions from existing information and data sets. To improve understanding and formalize the domain of complex containerized networking infrastructures, we can use knowledge management tools and specifications. An ontology-based approach for conceptualization and formal representation of domain knowledge allows the creation of knowledge graphs to denote and visualize elements of IaC networking domain. More precisely, we can express abstract knowledge uniquely by enriching the data with logic and semantics, making it both human and machine-understandable.

The objectives of this paper are to formally represent knowledge about core network connectivity concepts of multi-container applications and to integrate and validate networking policies based on the Description Logics (DLs) on data from real-world use cases, building on previous findings [4]. To show the potential of the ontology-based approach in semantically enriching the pre-deployment task of encoded infrastructure, we focus on the widely used platform for container configuration and management, utilizing Docker Compose orchestration files. Our proposed ontology, which formally describes the connectivity between containerized services, serves as the base for the knowledge graph generation. In addition, with the help of software reasoners, we validate the knowledge graphs' compliance with expert-defined rules and infer new information based on explicitly available knowledge.

The rest of this paper is organized as follows. Section II presents an overview of ontologies, knowledge graphs and their application in the networking domain. In Section III, we explain the semantically-enriched approach to verifying logic-based rules. Section IV evaluates this approach in 5G Core Network scenarios. In Section V, we provide a discussion of our findings, followed by the conclusion in Section VI.

978-3-903176-66-9

II. Related Work

Ontologies capture general knowledge while providing information share and reuse. Within the Information Technology (IT) field, an ontology is generally recognized as a "*formal, explicit specification of a shared conceptualization*" [5]. It includes formally defined concepts, their attributes, and relationships, collectively describing a phenomenon within a specific domain. An ontology can serve as the basis for knowledge graph construction, ensuring interoperability and containing real-world instances with well-defined properties.

We can logically formalize and model an ontology with the support of Description Logics (DLs) that allow the definition of concepts (classes), roles (binary relationships), and individuals (instances of classes) [6]. To make ontologies machine-interpretable, we can employ Semantic Web technologies that offer standardized methods for managing, storing, and querying data within knowledge graphs [7].

A well-known ontology, the TOUCAN ontology (ToCo) [8], captures physical components, users, services and metrics for channel quality assessment, focusing on hybrid telecommunication network systems. Compared to the ToCo ontology, the Infrastructure and Network Description Language (INDL) ontology [9], DevOps Infrastructure Ontology [10] and Container Description Ontology (CDO) [11] encompass the domain of virtualized computing infrastructures. However, while the focus of INDL is to describe storage and computing concepts, the DevOps Infrastructure Ontology is composed of 10 ontologies, each representing different aspects of an ICT system, from network infrastructure to organizational entities and business products and services. Furthermore, the CDO provides high-level definitions of core container orchestration concepts. However, it does not fully represent low-level networking concepts, allowing the description of the connectivity between services.

Within the domain of IaC, different methods for analyzing static configurations have been identified for detecting potential security weaknesses [12]. Kumara et al. [13] utilize Semantic Web technologies, creating the ontology representing the key application deployment concepts. Another approach employs dependency graphs created based on Ansible syntax using queries to detect inconsistencies [14]. Other proposed tools, such as the Docker Compose Validator [15] and Docker Composer [16], can assist developers in pre-deployment tasks, but they do not provide formal verification nor consider the semantics of networking in IaC, which can include important facts about the infrastructure.

The study of literature suggests that the application of ontologies have been explored to represent different concepts of virtualizing computing infrastructures and networking systems. Among them, the DevOps Infrastructure Ontology [10] covers core entities and their relationships, commonly described in configuration and IT service management databases. However, while the subsets of the DevOps Infrastructure Ontology address fundamental networking and virtualization concepts as well as the software-based nature of containers, they do not provide detailed descriptions of networking between containerized services. On the other hand, related work focused on IaC often disregards networking aspects and complexity. Inspired by these findings, we utilize the reusability principles to integrate the domains of networking and containerization and develop a formal model that represents network connectivity among containerized services.

III. Semantically-Enriched Containerization

In this section, we present an ontology that focuses on the core networking concepts of containerized services based on the previous findings [4]. Utilizing Docker Compose syntax, we populate the ontology, create a knowledge graph for querying existing information and automatically derive new insights using formal logic[1].

A. Container Networking Ontology

Adapting the *simple knowledge-engineering methodology* [17], we define the scope of our ontology through four main *competency questions*. With the first two questions, (1) how can two services (and their respective containers) be connected and (2) how can we ensure that two services are connected through the specific Internet Protocol (IP) network, we look into the network connectivity between containers. With the latter two questions, we investigate the representation of services and their networking attributes, (3) what type of IP address allocation a service can have, and (4) what type of port mapping is possible in the service definition.

The concepts of the `IP Address` and `IP Network` with a set of properties that can be useful to address our competency questions are defined in the DevOps Infrastructure Ontology [10], more precisely in an ontology covering network-related parts of the ICT infrastructure. In addition, the DevOps Infrastructure Ontology contains classes, such as `Virtual Server`, which could be extended to represent the concept of a container in a Docker Compose file. However, the `IP Network`, `IP Address` and `Virtual Server` classes are defined in different ontologies without a direct connection. Consequently, we create a Container Networking Ontology that builds upon the DevOps Infrastructure Ontology to represent networking among containerized services.

The resulting class hierarchy of our ontology is illustrated in Fig. 1. The classes `Configuration Item`, `Resource`, `IP Address` and `IP Network` are imported from the DevOps Infrastructure Ontology [10], while we define a new class `Service` which is an abstract definition of a Docker container.

The object and datatype properties of the Container Networking Ontology are illustrated in Fig. 2. The properties `belongs To IP Network` and `ip Address Version` are imported from the networking part of the DevOps Infrastructure Ontology. Each property has *rdfs:domain* and *rdfs:range* restrictions and *rdfs:label* and *rdfs:comment* for improved human readability.

[1]The knowledge graphs, ontology, parsing code, all the relevant data, and additional use cases are available at https://palmaitem.github.io/kgcompliance.

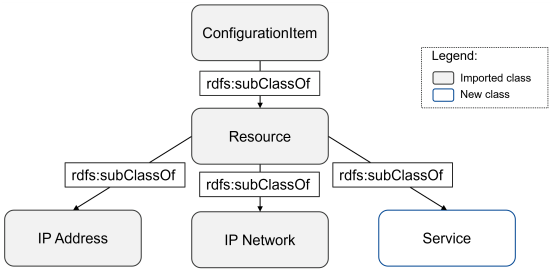

Fig. 1. The class hierarchy in the Container Networking Ontology.

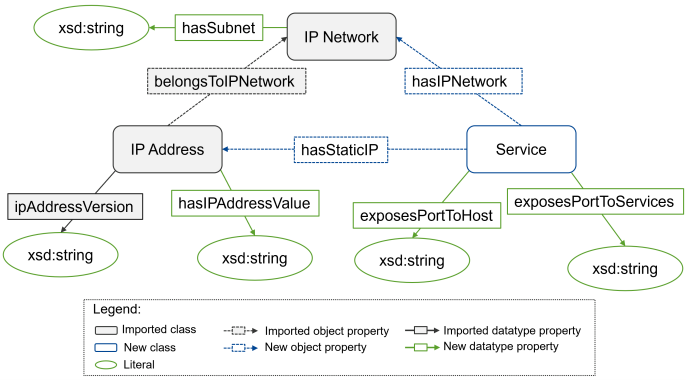

Fig. 2. Object and datatype properties in the Container Networking Ontology.

B. Knowledge Graph Creation

To create knowledge graphs with instances from real-world applications, we develop a parser that populates the Container Networking Ontology based on the Docker Compose syntax. This parsing module automatically creates a knowledge base having Container Networking Ontology in Extensible Markup Language (XML) format and a *compose.yaml* file as the input. With the help of reasoners integrated in *Protégé* [18], we also perform a consistency check to verify that each knowledge graph complies with the definition of concepts and their relationships in our ontology.

C. Inspecting User-defined Policies with Description Logics

Following the consistency check by reasoners applied to each knowledge graph using ontology-defined rules, we extend the graphs with our specified network policies. In this process, we add further semantics and demonstrate the potential for making conclusions on the encoded infrastructure based on formal logic.

The ontology-based approach allows various policies to be incorporated into the knowledge base as Semantic Web Rule Language (SWRL) rules. The defined rules are then parsed by the selected reasoner, automatically extending the knowledge base. The inferred knowledge, displayed as highlighted property assertions and queried with SPARQL Protocol and RDF Query Language (SPARQL), can help us to identify the causes of non-compliance.

Our defined rules are summarized in Table I. The first two logic-based rules universally inspect the network connectivity between containerized services. Rule 1 allows us to search for instances of the class `Service` connected to the default network. It is based on Docker Compose's default behaviour of creating a network that connects all the containers belonging to the same infrastructure when the connection between containers is not specified. Rule 2 explores different individuals of the class `Service` within the knowledge base that have the same `IP Network` defined, supposing that two containers within the same custom network should be able to communicate.

```
SELECT DISTINCT  ?service ?hostPort ?internalPort
WHERE {
?service rdf:type :Service .
?service :nonCompliance true.
{?service :exposesPortToHost ?hostPort} UNION
{?service :exposesPortToServices ?internalPort} .
}
```

Listing 1. SPARQL query to identify non-compliant services with Rule 4 and their exposed ports.

Given the widespread use of Hypertext Transfer Protocol (HTTP) traffic in modern applications and 5G networks, we examine the exposure of its default ports. The following two rules are based on the standard protocol-port mapping, assuming that the security measure allows HTTP and HTTP Secure (HTTPS) traffic over ports 80 and 443, respectively. Therefore, Rule 3 inspects if default network port 80 is exposed to the Docker Host. Rule 4 verifies port 443 exposure to the Docker Host, which would be a typical compliance control for ensuring that HTTP external traffic in the Docker network is encrypted. This rule also checks if internal traffic is limited to port 80, typically used by modern services' APIs. However, these protocol-related rules could include other ports approved by a network security team.

The knowledge derived from evaluating the defined rules is stored within the analyzed knowledge graph and queried to assess the compliance of each knowledge base with the specified network policies. For instance, the SPARQL query presented in Listing 1 allows us to determine whether the standard HTTP ports are exposed externally and, therefore, potentially to public networks or internally only to other Docker containers.

IV. Automated Validation of the 5G Core Network Configuration Files

To demonstrate the potential of the ontology-based approach to represent containerized network infrastructures and verify user-defined network policies, we present the validation of two knowledge graphs. These graphs are automatically generated based on real-world data parsed from open-source 5G Core Network deployment scenarios and checked for compliance with our defined rules.

Table II summarizes the extracted knowledge for the two use cases presented in more detail in sections IV-A and IV-B. Each 5G Core architecture has a different number of instances of the class `Service` defined by our Container Networking Ontology, presented as *Total Services*, and distinct network configurations, despite both infrastructures serving the same

TABLE I
NETWORK POLICIES DEFINED AS SWRL RULES.

Rule	Description
1. Default Compose Network	`Service(?x) ∧ hasIPNetwork(?x,?y) → hasDefaultIPNetwork(?x, false)`
2. Connectivity between Services	`Service(?x) ∧ Service(?y) ∧ hasIPNetwork(?x,?s) ∧ hasIPNetwork(?y,?s) ∧ differentFrom(?x,?y) → hasConnectivityWith(?x,?y)`
3. Exposed HTTP Port	`Service(?x) ∧ exposesPortToHost(?x,?p) ∧ swrlb:equal(?p,"80") → exposesHTTPPortExternally(?x,true)` `Service(?x) ∧ exposesPortToHost(?x,?p) ∧ swrlb:equal(?p,"80/tcp") → exposesHTTPPortExternally(?x,true)`
4. Compliance Check	`Service(?x) ∧ exposesPortToHost(?x,?p) ∧ swrlb:notequal(?p, "443") ∧ swrlb:notequal(?p,"443/tcp") → nonCompliance(?x,true)` `Service(?x) ∧ exposesPortToServices(?x,?p) ∧ swrlb:notequal(?p, "80") ∧ swrlb:notequal(?p,"80/tcp") → nonCompliance(?x,true)`

purpose. The row *Custom IP Network* represents the number of services satisfying Rule 1, revealing that these deployments do not rely on Docker's default network, while the row *HTTP Host Ports* presents the outcomes of validating Rule 3. *HTTP Internal Ports* provides the number of standard HTTP ports exposed internally between services, while the row *Non-compliant Services* represents the automatically inferred knowledge after the integration of Rule 4. The remaining results include *Host Ports* and *Internal Ports*, which contain the total number of exposed ports. In addition, the *# IP Networks* and *# IP Addresses* show that while both 5G cores define a single IP network, OAI5GC is more explicit in the definition of addresses.

TABLE II
EXTRACTED KNOWLEDGE USING SPARQL QUERIES.

	Free5GC	**OAI5GC**
Total Services	15	9
Custom IP Network	15	9
HTTP Host Ports	0	0
HTTP Internal Ports	0	6
Non-compliant Services	11	7
Host Ports	3	0
Internal Ports	12	16
# IP Networks	1	1
# IP Addresses	1	9

A. Free 5G Core Solution

Based on the publicly available Docker Compose file[2] created as part of the Free5GC project, we build a knowledge graph named Free5GC. It contains 15 instances of the class `Service`, while there is only one instance of `IP Network` and `IP Address` classes, as presented in Table II. This indicates that Rule 2 is consistently applied to all services and that they share the same network definition. Furthermore, no specific default HTTP ports are detected, implying that the 11 Services that do not satisfy Rule 4 arise from exposure of ports other than 80 and 443.

B. Open Air Interface 5G Core Network Solution

Another 5G Core Network deployment, used to populate our ontology and validate network policies, is provided as part of the OpenAirInterface Software Alliance (OAI) project. To build the OAI5GC knowledge graph, we refer to the *Basic 5GC* deployment mode[3].

There is a total of nine instances for each of the classes `Service` and `IP Address`, with a single `IP Network` defined, as presented in Table II. The result for the *Custom IP Network* suggests that all the services connect to the same custom network. Furthermore, there are 16 *Internal Ports* defined, out of which six are *HTTP Internal Ports*, while seven services do not satisfy Rule 4. Thus, we can conclude that other ports than 80 are exposed internally. This inferred knowledge can further be explored with various SPARQL queries and assist network developers in gaining a deeper understanding and potentially modifying our defined rules to automate the verification of their configurations.

V. DISCUSSION

The Container Networking Ontology is constructed based on the four main competency questions, and it does not include the definition of Docker-specific attributes. It represents the knowledge of networking between containerized services beyond Docker. Therefore, the proposed semantically-enriched containerization approach has the potential for extension and modification to include more networking options available in different IaC specifications.

The parser responsible for knowledge graph creation can be adapted to be agnostic to the underlying container-based architecture. This allows further enhancement and validation of more complex deployment scenarios utilizing various container orchestration systems, such as Kubernetes. To extend our prototype beyond the Docker Compose syntax, we could

[2]*docker-compose.yaml* available at github.com/free5gc/free5gc-compose/tree/master, with the commit hash f37df73.

[3]*docker-compose-basic-nrf.yaml* available at gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed/-/tree/master/docker-compose, with the commit hash 8848fde.

use annotations in the defined IaC files and enable more explicit mapping of concepts and properties.

The proposed validation of configuration files currently assists in adding semantic value only to static definitions within the pre-deployment tasks. However, the scope of our model can be broadened to cover the operation, management and maintenance of a typical infrastructure. For example, by keeping an up-to-date knowledge base through monitoring agents, compliance checks against defined logic rules can be performed in real time while the application is running. By tracking changes, autonomic and cognitive management could be achieved, where we could detect misconfigurations — assisted by the reasoning features of our model — and gain a deeper understanding of the sources for non-compliance. Potentially, this approach could even be used to prevent changes that would cause issues or network security risks, requiring approval or validation from an expert.

The semantically-enriched containerization may help human actors by creating a common understanding of the system, which could improve knowledge sharing and overall interoperability. Therefore, it could be adopted within ICT organizations to ease not only human-to-machine but also human-to-human communication. Furthermore, our prototype could be improved with human-centered evaluation, providing the domain knowledge for Artificial Intelligence (AI) models to achieve more comprehensible and intelligent network and service configuration and management.

VI. Conclusion

This paper demonstrates an approach for analyzing IaC deployments through a human-readable definition of concepts and their relations, represented in a flexible ontology. By enriching the available configurations with explicit semantics, we can automatically employ user-defined networking policies through formal logic. These policies can be represented universally (i.e., mapped to well-defined concepts) and be transparently applied to IaC without requiring any modification, allowing machine-based interpretation and reasoning of the parsed IaC data. We validate our approach in two reference 5G Core Network implementations and show how automatically inferred insights can assist the diagnostic operations by detecting non-compliant configurations. This approach can serve as a basis for human actors to make more informed decisions, which can become an integral part of the life cycle of a cloud-native infrastructure. It also allows the development of a common understanding between various human actors who can evaluate their network and security policies by inspecting different IaC deployments with our proposed ontology.

References

[1] T. Wood, K. K. Ramakrishnan, J. Hwang, G. Liu, and W. Zhang, "Toward a software-based network: integrating software defined networking and network function virtualization," *IEEE Network*, vol. 29, no. 3, pp. 36–41, 2015.

[2] K. Morris, *Infrastructure as Code*. O'Reilly Media, Inc., Dec. 2020.

[3] W. Attaoui, E. Sabir, H. Elbiaze, and M. Guizani, "VNF and CNF Placement in 5G: Recent Advances and Future Trends," *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 4698–4733, Dec. 2023.

[4] A. Simić, "Speaking the same language through logic and ontologies," Master's thesis, Norwegian University of Science and Technology, 2024.

[5] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge engineering: Principles and methods," *Data & Knowledge Engineering*, vol. 25, no. 1, pp. 161–197, Mar. 1998.

[6] S. Rudolph, "Foundations of Description Logics," in *Reasoning Web. Semantic Technologies for the Web of Data*, A. Polleres, C. d'Amato, M. Arenas, S. Handschuh, P. Kroner, S. Ossowski, and P. Patel-Schneider, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 6848, pp. 76–136, series Title: Lecture Notes in Computer Science.

[7] D. Allemang, J. Hendler, and F. Gandon, *Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL*, 3rd ed. New York, NY, USA: Association for Computing Machinery, 2020, vol. 33.

[8] Q. Zhou, A. J. G. Gray, and S. McLaughlin, "ToCo: An Ontology for Representing Hybrid Telecommunication Networks," in *The Semantic Web*. Cham: Springer International Publishing, 2019, vol. 11503, pp. 507–522, series Title: Lecture Notes in Computer Science.

[9] M. Ghijsen, J. van der Ham, P. Grosso, C. Dumitru, H. Zhu, Z. Zhao, and C. de Laat, "A semantic-web approach for modeling computing infrastructures," *Computers & Electrical Engineering*, vol. 39, no. 8, pp. 2553–2565, Nov. 2013.

[10] O. Corcho, D. Chaves-Fraga, J. Toledo, J. Arenas-Guerrero, C. Badenes-Olmedo, M. Wang, H. Peng, N. Burrett, J. Mora, and P. Zhang, "A High-Level Ontology Network for ICT Infrastructures," in *The Semantic Web – ISWC 2021*, ser. Lecture Notes in Computer Science, A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P. Barnaghi, A. Haller, M. Dragoni, and H. Alani, Eds. Cham: Springer International Publishing, 2021, pp. 446–462.

[11] K. Boukadi, M. Rekik, J. B. Bernabe, and J. Lloret, "Container description ontology for CaaS," *International Journal of Web and Grid Services*, vol. 16, no. 4, pp. 341–363, Jan. 2020.

[12] P. R. Reddy Konala, V. Kumar, and D. Bainbridge, "SoK: Static Configuration Analysis in Infrastructure as Code Scripts," in *2023 IEEE International Conference on Cyber Security and Resilience (CSR)*, Jul. 2023, pp. 281–288.

[13] I. Kumara, Z. Vasileiou, G. Meditskos, D. A. Tamburri, W.-J. Van Den Heuvel, A. Karakostas, S. Vrochidis, and I. Kompatsiaris, "Towards Semantic Detection of Smells in Cloud Infrastructure Code," in *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*. Biarritz France: ACM, Jun. 2020, pp. 63–67.

[14] R. Opdebeeck, A. Zerouali, and C. De Roover, "Control and Data Flow in Security Smell Detection for Infrastructure as Code: Is It Worth the Effort?" in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, May 2023, pp. 534–545, iSSN: 2574-3864.

[15] (2019) Introducing the Docker Compose Validator. [Online]. Available: https://blog.zhaw.ch/splab/2019/10/04/introducing-the-docker-compose-validator/

[16] B. Piedade, J. P. Dias, and F. F. Correia, "Visual notations in container orchestrations: an empirical study with Docker Compose," *Software and Systems Modeling*, vol. 21, no. 5, pp. 1983–2005, Oct. 2022.

[17] N. F. Noy and D. L. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Last Access: Jan. 01, 2024. [Online]. Available: https://protege.stanford.edu/publications/ontology_development/ontology101.pdf

[18] M. A. Musen, "The Protégé Project: A Look Back and a Look Forward," *AI matters*, vol. 1, no. 4, pp. 4–12, Jun. 2015.