Representação Baseada em Grafos de Infraestrutura como Código: Possibilitando o Raciocínio Semântico para Sistemas Conteinerizados

Guilherme M. Soares[1], **Lucas S. Vrielink**[1], **Juliano A. Wickboldt**[1], **Jéferson C. Nobre**[1], **Lisandro Z. Granville**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brasil

{gmsoares, lsvrielink, jwickboldt, jcnobre, granville}@inf.ufrgs.br

Abstract. *This paper proposes a new framework that integrates an ontology-based Knowledge Graph with the Model Context Protocol (MCP) for the semantic management of containerized infrastructures. The framework addresses a gap in orchestration platforms by providing a unified semantic representation of the topology, powered by an Extraction, Transformation, and Loading (ETL) process that analyzes Docker Compose files and makes the knowledge available to Large Language Models via a tool layer. This approach enables users to perform natural language queries to obtain deep insights into infrastructure configuration. The approach was validated through a controlled experiment simulating a DevOps scenario, in which the system successfully identified volume integrity failures and lateral movement risks—issues that traditional analysis would likely struggle to detect. The results demonstrate the framework's superiority in automated auditing and conflict detection, bridging the gap between raw Infrastructure-as-Code artifacts and dynamic system reasoning for more intelligent and explainable container management.*

Resumo. *Este artigo propõe um novo framework que integra um Grafo de Conhecimento baseado em ontologia com o Model Context Protocol (MCP) para o gerenciamento semântico de infraestruturas conteinerizadas. A estrutura aborda uma lacuna nas plataformas de orquestração ao fornecer uma representação semântica unificada da topologia, alimentada por processo de Extração, Transformação e Carga (ETL) que analisa arquivos do Docker Compose e disponibiliza o conhecimento para Large Language Models via uma camada de ferramentas. Essa abordagem permite que usuários realizem consultas em linguagem natural para obter insights profundos sobre a configuração da infraestrutura. A abordagem foi validada por meio de um experimento controlado simulando um cenário DevOps, no qual o sistema identificou com sucesso falhas de integridade em volumes e riscos de movimentação lateral — problemas que a análise tradicional dificilmente detectaria. Os resultados demonstram a superioridade da estrutura na auditoria automatizada e na detecção de conflitos, preenchendo a lacuna entre artefatos brutos de Infraestrutura como Código e o raciocínio dinâmico do sistema para um gerenciamento de contêineres mais inteligente e explicável.*

1. Introdução

A virtualização baseada em contêineres se consolidou na computação em nuvem moderna, oferecendo uma alternativa eficiente às máquinas virtuais tradicionais [Kerner 2025]. De acordo com um relatório emitido pela empresa *Nutanix* em 2025 [Nutanix 2025], cerca de 90% das empresas consultadas informam utilizar aplicações containerizadas em alguma capacidade. No entanto, essa adoção massiva traz consigo desafios de gerenciamento e compreensão da infraestrutura. Conforme destacado por [Zhou et al. 2020] ao propor o *DockerKG*, a complexidade das relações entre os artefatos dos contêineres exige modelos de dados semanticamente ricos, como grafos de conhecimento, para organizar e compreender as conexões entre entidades que, de outra forma, ficariam dispersas ou isoladas.

A conteinerização facilita o empacotamento de aplicações e suas dependências, garantindo portabilidade entre ambientes distintos. Com a expansão das arquiteturas de microserviços, a necessidade de gerenciar de forma eficiente inúmeros recursos levou à adoção de ferramentas de orquestração. Nesse contexto, plataformas como o *Kubernetes* tornaram-se fundamentais para automatizar a implantação e o gerenciamento do ciclo de vida desses serviços. No entanto, essa evolução moveu a gestão de infraestrutura para o paradigma *Infrastructure as Code* (IaC). Conforme observado por [Simić and Palma 2024], embora esses artefatos facilitem a automação, eles possuem um importante valor semântico que muitas vezes permanece subutilizado, funcionando apenas como instruções estáticas de baixo nível em vez de uma base de conhecimento compreensível.

A automação, embora robusta, ainda tem uma limitação fundamental: ela executa tarefas, mas não consegue analisar o cenário completo e tomar decisões inteligentes com base no estado do sistema. As ferramentas tradicionalmente utilizadas para este tipo de verificação, como *linters* estáticos e CLI's focam apenas nos contêineres que estão sendo executados e sintaxes dos arquivos de definição, mas falham em fornecer uma compreensão de como os componentes interagem. Embora trabalhos recentes tenham explorado o uso de grafos para conformidade de rede [Simić and Palma 2024] ou ontologias para descrição de serviços [Boukadi et al. 2020], ainda existe uma lacuna: a falta de uma interface unificada que integre a análise estática com um raciocínio dinâmico. As soluções existentes tendem a ser restritas a domínios específicos ou não oferecem mecanismos acessíveis, como linguagem natural, para que operadores auditem dependências transitivas e riscos de segurança em arquiteturas complexas.

O objetivo deste artigo é preencher essa lacuna, com uma metodologia que conecta um grafo de conhecimento semanticamente rico — modelando a topologia do contêiner e suas propriedades — a um Agente de Infraestrutura. Essa conexão cria um plano de gerenciamento unificado, permitindo que usuários realizem consultas em linguagem natural para obter *insights* sobre relacionamentos de rede, dependências de recursos e integridade, sem a necessidade de inspeção manual de arquivos de configuração de baixo nível.

A eficácia da abordagem proposta foi validada por meio de experimentos controlados simulando cenários de DevOps. Os resultados demonstram a melhora da estrutura na detecção de problemas que escapam à análise estática tradicional. O sistema identificou com sucesso cadeias complexas de dependência, conflitos de integridade de volumes e vulnerabilidades de movimentação lateral na rede. Além disso, a centralização das

informações no grafo reduziu a carga cognitiva operacional, transformando verificações manuais demoradas em consultas de tempo constante.

Essa conexão baseada em Model Context Protocol (MCP) cria um plano de gerenciamento unificado em nossa proposta. Ela permite que usuários e agentes de IA realizem consultas estruturadas ou em linguagem natural no Grafo de Conhecimento para obter *insights* sobre contêineres em execução — como seus relacionamentos de rede, dependências de recursos e *status* de integridade em tempo real — sem a necessidade de acessar ou analisar diretamente arquivos de configuração de baixo nível. Essa camada semântica, viabilizada pelo MCP, torna o gerenciamento e a orquestração de contêineres mais inteligentes, explicáveis e acessíveis.

Este artigo está organizado da seguinte forma. A seção 2 explica melhor alguns dos conceitos básicos relacionados a este trabalho. A Seção 3 apresenta um panorama dos trabalhos relacionados. A Seção 4 descreve a metodologia proposta. A Seção 5 apresenta e discute os resultados obtidos. A Seção 6 resume os principais resultados deste trabalho. Finalmente, a Seção 7 apresenta os próximos passos para o aprimoramento deste projeto.

2. Fundamentos

Nesta seção são apresentados alguns conceitos importantes para o entendimento do trabalho proposto. Aqui, é explicado o que são grafos de conhecimento, ontologias, Large Language Models (LLMs), MCP *Tools* e como essas tecnologias são combinadas neste trabalho.

2.1. Grafos de Conhecimento

Introduzido inicialmente pelo *Google* em 2012 [Singhal 2012], um grafo de conhecimento, também conhecido como rede semântica, é um modelo de dados semanticamente rico para armazenar, organizar e compreender entidades conectadas [Claise 2025]. A unidade básica de um grafo de conhecimento é um conjunto de tripletos. Um tripleto é representado como (entidade, relação, entidade), contendo as entidades e as relações entre elas. As entidades em um grafo de conhecimento são conectadas por meio de relações entre elas, formando uma estrutura de conhecimento em rede [Zhou et al. 2020].

2.2. Ontologias

Em ciência da computação, uma ontologia é "uma especificação formal e explícita de uma conceitualização compartilhada para um domínio de interesse" [Gruber 1993]. Uma "conceitualização" refere-se a um modelo abstrato de algum fenômeno no mundo por meio da identificação dos conceitos relevantes desse fenômeno [Studer et al. 1998]. A ontologia é "explícita" porque define claramente os tipos de conceitos e as regras de uso, é "formal", pois deve ser legível por máquina, o que exclui a linguagem natural, e é "compartilhada", pois representa um conhecimento consensual aceito por um grupo.

As ontologias são basicamente estruturas de conhecimento de um domínio específico que permitem que os sistemas processem automaticamente o significado das informações, deduzam informações implícitas a partir das explícitas e integrem bancos de dados heterogêneos [Kyriakopoulos et al. 2015].

2.3. *LLMs*

LLMs são sistemas avançados de IA que compreendem e geram linguagem natural, ou texto semelhante ao humano, usando os dados com os quais foram treinados por meio de técnicas de *machine learning*. Os LLMs podem gerar automaticamente conteúdo baseado em texto, que pode ser aplicado a uma ampla gama de casos de uso em setores como saúde [Shool et al. 2025], telecomunicações [Zhou et al. 2025], cibersegurança [Yao et al. 2024] e finanças [Wu et al. 2023]. Exemplos incluem, GPT, Claude, e Deepseek.

LLMs representam um grande avanço na forma como os humanos interagem com a tecnologia, pois são o primeiro sistema de IA capaz de lidar com linguagem humana não estruturada em larga escala, permitindo uma comunicação natural com as máquinas [IBM and Stryker 2023]. No entanto, esta tecnologia ainda apresenta problemas, como alto consumo de eletricidade, alucinações, falta de transparência em seu funcionamento e tendenciosidade.

É possível conectar uma LLM à dados externos através de opções como a *Agentic AI*. A forma como os agentes se comunicam com as ferramentas envolve orquestração, com fluxos ou grafos dependendo da estrutura utilizada. Essa abordagem permite que o LLM "raciocine" e determine a melhor maneira de responder a uma pergunta — como decidir se a consulta pode ser respondida com as informações disponíveis ou se uma pesquisa externa é necessária [Red Hat 2025].

2.4. Docker

O *Docker* se consolidou como uma ferramenta padrão para conteinerização, permitindo o encapsulamento de aplicações e suas dependências em unidades leves e portáteis que garantem a execução consistente em ambientes heterogêneos. Para lidar com a orquestração de arquiteturas com múltiplos contêineres, o *Docker Compose* fornece um nível mais alto de abstração, utilizando um arquivo YAML declarativo para definir a topologia do sistema. Este arquivo de configuração serve como um modelo para a infraestrutura, especificando rigorosamente serviços, redes e volumes de armazenamento persistente, bem como suas interdependências e configurações de tempo de execução, permitindo que sistemas distribuídos complexos sejam provisionados e gerenciados como uma única entidade coesa [Docker 2025].

2.5. Model Context Protocol

O MCP fornece uma maneira padronizada de desacoplar LLMs e agentes de IA de ferramentas e fontes de dados específicas. Ele atua como um adaptador universal, permitindo que uma IA interaja com sistemas externos — de bancos de dados e APIs a ferramentas de software — de maneira estruturada e consistente, possibilitando assim a automação de fluxos de trabalho complexos. A aplicação do MCP para gerenciamento de contêineres e infraestrutura visa refatorar as capacidades operacionais como "ferramentas" detectáveis, proporcionando uma arquitetura mais ágil e extensível para a integração de IA.

O MCP define uma camada de comunicação unificada que suporta interações *stateful* entre três funções: o *Host* MCP, o Cliente MCP e o servidor MCP. O *Host* MCP orquestra a execução do fluxo de trabalho e intermedia as interações entre o LLM, o Cliente e o Servidor. O Servidor MCP gerencia o acesso a recursos de dados estruturados e

executa ações por meio de Ferramentas, atuando como uma ponte para ecossistemas externos. As comunicações são realizadas por meio de solicitações, respostas e notificações JSON-RPC [Wang et al. 2026].

3. Trabalhos Relacionados

A gestão de infraestrutura evoluiu de *scripts* imperativos para o paradigma de Infraestrutura como Código (IaC). No entanto, a crescente complexidade dessas arquiteturas revela uma lacuna nos modelos que vai além de uma simples análise de arquivos de configuração. Os trabalhos revisados a seguir abordam esse problema de complexidade no IaC ou apresentam ferramentas e conceitos que podem ser úteis para a metodologia aqui proposta. O objetivo desta seção é, portanto, apresentar essas contribuições e destacar tanto os pontos de divergência quanto os de convergência entre elas e a metodologia deste artigo.

Em [Simić and Palma 2024], os autores propõem uma abordagem que utiliza um Grafo de Conhecimento, povoado com políticas de rede, para automatizar a verificação de *compliance* em serviços conteinerizados. Simic e Palma analisam as configurações de rede por meio de um grafo baseado em ontologia, o que permite a verificação automatizada de conectividade e regras de política, tarefa de difícil execução quando realizada diretamente sobre arquivos de configuração brutos. Ao aplicar o método ao domínio específico da *compliance* de rede, o estudo exemplifica como a transformação de artefatos de IaC em Grafos de Conhecimento baseados em ontologias pode enriquecer semanticamente tais artefatos. Nossa metodologia usa também Grafos de Conhecimento, porém com foco na topologia de contêineres e itera sobre ele ao adicionar interatividade via um Agente de Infraestrutura.

No contexto do uso de ontologias para apoiar ambientes baseados em contêineres, [Boukadi et al. 2020] propõem uma *Container Description Ontology* (CDO), uma ontologia de domínio projetada para descrever semanticamente contêineres, *Docker* e sistemas de orquestração de contêineres no contexto de *Containers as a Service* (CaaS). O CDO modela de forma abrangente as capacidades funcionais e não funcionais desses componentes, visando abordar o desafio de descobrir e selecionar ferramentas de orquestração de contêineres apropriadas em meio a um cenário heterogêneo. Os autores também apresentam uma estrutura de suporte que facilita a população da ontologia a partir de fontes como o *Docker Hub*, permite a migração de contêineres entre diferentes plataformas de nuvem para interoperabilidade e auxilia os usuários na descoberta de sistemas de orquestração adequados por meio de regras de inferência baseadas em capacidades definidas na Linguagem de Regras da Web Semântica (SWRL). Nossa metodologia também atua no domínio da orquestração de contêineres, porém difere ao adicionar a interatividade disponibilizada pelo MCP.

Na comunidade do *Internet Engineering Task Force* (IETF), o grupo de trabalho de Operações de Gerenciamento de Rede (NMOP) tem explorado a integração de tecnologias semânticas com linguagens de modelagem padrão. Um esforço recente é o trabalho de [Martinez-Casanueva 2025], que propõe uma estrutura formal para mapear modelos de dados YANG em uma estrutura de grafo de conhecimento, visando validar a integridade da configuração da rede. Enquanto essa proposta foca na validação de baixo nível de protocolos de rede, nosso trabalho eleva a abstração para a camada de aplicação (con-

têineres), democratizando o acesso de validações complexas por meio de uma interface conversacional.

Uma ontologia para capturar o conhecimento do domínio de IaC e fornecer assistência interativa, como autocompletar modelos, validação, detecção de *code smells* (ruídos no código) e correspondência de recursos é proposta em [Vasileiou et al. 2025]. Uma *engine* é responsável por traduzir os modelos para a ontologia e armazená-los na base de conhecimento, onde um raciocinador semântico é empregado para fornecer orientações com base em consultas SPARQL. Contudo, a dependência de consultas SPARQL limita a usabilidade de operadores comuns. Nossa solução supera esta barreira ao utilizar uma camada de ferramentas para abstrair a intenção do usuário em linguagem natural diretamente para operações determinísticas no grafo.

A literatura recente aponta a integração entre LLMs e grafos de conhecimento como uma estratégia para mitigar alucinações e enriquecer o contexto de modelos generativos [Pan et al. 2024]. Em domínios de alta precisão, como DevOps, a exatidão de identificadores de contêineres e portas é mandatória, logo, a dependência exclusiva do treinamento prévio do modelo se torna um risco operacional. Nossa proposta materializa esse conceito teórico especificamente no domínio Docker, implementando uma metodologia onde o grafo atua como fonte da verdade obrigatória para as respostas do agente, eliminando a possibilidade de alucinações inerente à geração de texto livre.

Por fim, em [Wang et al. 2026] é apresentada uma implementação sofisticada de um padrão arquitetural semelhante ao que propusemos, mas especializada para a área de ataque de testes de penetração automatizados. O sistema *PTFusion* utiliza MCP, grafo de conhecimento e agentes para modelar e atacar ecossistemas, usando agentes "adversários" para planejar caminhos de exploração. Embora a arquitetura base seja similar, nosso trabalho inverte a lógica utilizando o raciocínio semântico para garantir a integridade, conformidade e estabilidade da implantação em vez de utilizar para ataques de ecossistemas, provando a versatilidade da combinação do MCP e grafos de conhecimento para diferentes fins operacionais.

4. Proposta

Esta seção descreve a metodologia utilizada para o gerenciamento semântico de infraestruturas baseadas em contêineres. A metodologia aborda as seguintes propriedades: (1) a formalização ontológica dos conceitos de infraestrutura do *Docker*; (2) o desenvolvimento de um analisador sintático para ingerir dados no grafo de conhecimento com base em uma ontologia previamente definida; (3) a implementação de um servidor MCP para raciocínio LLM e interação com linguagem natural. O desenvolvimento completo, incluindo código, experimentos, prompts e detalhes de implementação, pode ser encontrado no repositório do projeto no GitHub[1].

4.1. Visão Geral da Metodologia

A metodologia proposta segue um fluxo de Extração, Transformação e Carga (ETL), culminando em uma camada de inferência semântica. Conforme ilustrado na Figura 1, o sistema consome os arquivos de definição de infraestrutura (Artefatos de IaC), os processa por meio de um analisador baseado em ontologia e armazena as informações em um

[1] `https://github.com/ComputerNetworks-UFRGS/compose-kg-mcp`

banco de dados de grafos. Posteriormente, uma camada de abstração expõe ferramentas semânticas que permitem aos agentes de IA consultar e raciocinar sobre as características dos artefatos de IaC. Essa abordagem assegura um fluxo de dados robusto e garante que o raciocínio do sistema seja fundamentado em uma base de conhecimento consistente, mitigando alucinações.

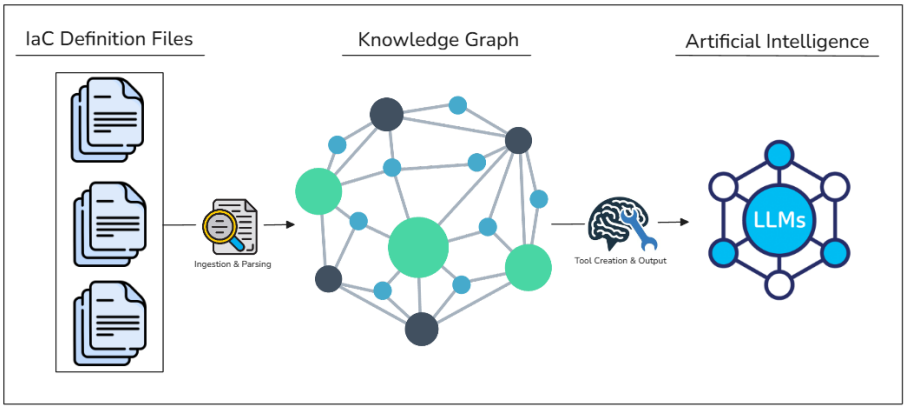

Figura 1. Visão Geral da Metodologia

4.2. Formalização Ontológica para Containers Docker

Uma ontologia foi desenvolvida utilizando a *Ontology Web Language* (OWL) para obter uma representação semântica padronizada das características dos arquivos de definição do Docker. A OWL permite representar formalmente o conhecimento sobre classes, propriedades e relacionamentos, viabilizando inferências lógicas.

A taxonomia proposta dos componentes de orquestração de contêineres garante a integridade do modelo, facilitando a validação lógica das dependências. A Figura 2 apresenta a representação esquemática das classes e relações.

A classe `Service` atua como unidade central, representando o contêiner ou microsserviço. A classe `Network` define a topologia de comunicação, permitindo a análise de conectividade, enquanto *Volume* aborda a camada de persistência de dados. A interface de comunicação entre o *host* e o contêiner é estabelecida pela classe `PortMapping`. As configurações de tempo de execução, por sua vez, são encapsuladas pela classe `EnvironmentVariable`. Por fim, a classe `Image` define o modelo base do serviço.

As relações foram modeladas para capturar as dependências funcionais e estruturais: *DEPENDS_ON* define a hierarquia de inicialização; *CONNECTS_TO* mapeia a topologia da rede; *EXPOSES_PORT* define a superfície de comunicação; *USES_IMAGE* vincula o serviço ao seu artefato binário; *HAS_ENV_VAR* e *MOUNTS_VOLUME* definem as variáveis de ambiente e os volumes, respectivamente, instanciados no serviço.

4.3. Ingestão do Grafo de Conhecimento

A construção do grafo se baseia em um módulo estático responsável por interpretar os arquivos de definição de infraestrutura. Dada a natureza dessas definições, onde os serviços podem referenciar dependências declaradas posteriormente no arquivo, o processo de ingestão utiliza uma estratégia de duas fases.

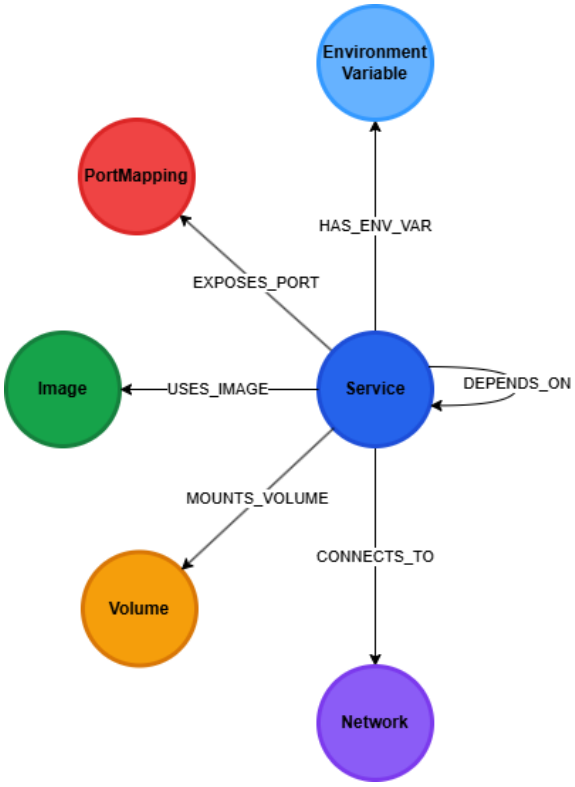

Figura 2. Grafo da Ontologia

A primeira fase identifica e cria todos os nós de serviço com seus Identificadores Uniformes de Recursos (URIs). Esta etapa assegura a existência de todas as entidades no grafo antes que quaisquer relacionamentos sejam aplicados. Logo, a segunda etapa processa todas as propriedades detalhadas e estabelece as arestas de relacionamento entre as entidades já existentes, garantindo a consistência referencial da topologia.

4.4. Raciocínio Semântico e Camada de Ferramentas

A interação em linguagem natural é viabilizada por uma camada de abstração de ferramentas. Ao contrário de abordagens que instruem LLMs a gerar consultas de banco de dados diretamente (propensas a erros de sintaxe e alucinações de esquema), esta metodologia emprega um protocolo de ferramentas determinísticas.

O sistema expõe funções que abstraem a complexidade da linguagem de consulta ao grafo. O modelo de linguagem atua, portanto, como um orquestrador de raciocínio, ou seja, ele interpreta a intenção do usuário e decide qual ferramenta invocar. O Grafo de Conhecimento, acessado por essas ferramentas, fornece a verdade factual sobre a infraestrutura, fundamentando as respostas geradas.

4.5. Aspectos da Implementação

Para validar a metodologia proposta, desenvolveu-se um protótipo funcional utilizando tecnologias de mercado. A ingestão foi implementada em *Python*, focada na análise sintática de arquivos YAML do *Docker Compose*. O armazenamento semântico utiliza o banco de dados orientado a grafos *Neo4j*.

A camada de ferramentas foi desenvolvida através do MCP. O servidor MCP implementado expõe capacidades específicas tais como: rastreamento de dependências transitivas entre os serviços; inspeção do isolamento de redes; e detecção de integridade dos volumes dos containers. Essa implementação desacopla o agente de IA da base de dados, utilizando o MCP como interface padronizada de troca de contexto.

5. Resultados

A avaliação da estrutura proposta foi conduzida sob duas perspectivas complementares: uma análise quantitativa, focada na eficiência computacional e na densidade da representação semântica; e uma análise qualitativa baseada em estudos de caso, demonstrando a capacidade de raciocínio sobre problemas complexos de topologia e segurança.

5.1. Cenário Experimental

Os experimentos foram executados em um computador equipado com um processador Intel Core i5, 16 GB de RAM e um ambiente Docker isolado, ou seja, apenas com containeres dos arquivos *docker-compose* relacionados ao artigo. O teste consistiu em um conjunto sintético de arquivos *docker-compose*, variando de arquiteturas monolíticas simples a microsserviços distribuídos, desenhados para conter falhas de configuração intencionais (por exemplo, conflitos de recursos, violações de segmentação). As funções do MCP foram testadas a partir do cliente *Claude* com o modelo *Sonnet 4.5*.

5.2. Análise Quantitativa

Uma das premissas deste trabalho é que a transformação de código estático em um grafo de conhecimento enriquece a representação da infraestrutura. Conforme demonstrado na Figura 3, observa-se uma progressão linear entre o volume de código e a estrutura de grafo gerado. Essa análise se inicia no `case_4`, o cenário mais simples com 25 linhas de código resultado em 20 entidades, que se estende até o `case_1`, com 58 linhas gerando 43 entidades. O dado crucial aqui é o número de entidades do grafo (nós e arestas) permanece sistematicamente inferior à contagem de linhas de código. Isso valida a capacidade da ingestão filtrar o ruído da sintaxe dos arquivos (chaves do YAML, indentação e metadados vazios) e transformar apenas a essência arquitetural ao grafo, possibilitando uma compreensão semântica sem perda de dados funcionais.

Além da eficiência de armazenamento, os dados evidenciam que a complexidade do grafo escala de maneira sustentável e previsível. Ao analisar a transição do `case_2` (29 linhas para 29 entidades) para o `case_3` (39 linhas para 30 entidades), se nota que o aumento no código não gera uma explosão combinatória de nós, mas sim um enriquecimento da topologia. A vantagem operacional reside na natureza dessas entidades: enquanto 58 linhas de código do `case_1` exigem uma leitura sequencial e cognitiva para correlacionar os serviços, as 43 entidades representam de forma simples as conexões de pontos. Ou seja, a ingestão dos arquivos para o grafo de conhecimento permite que a arquitetura seja atravessada de ponta a ponta com o custo computacional reduzido, independente se o ambiente possui a simplicidade do `case_4` ou a densidade do `case_1`.

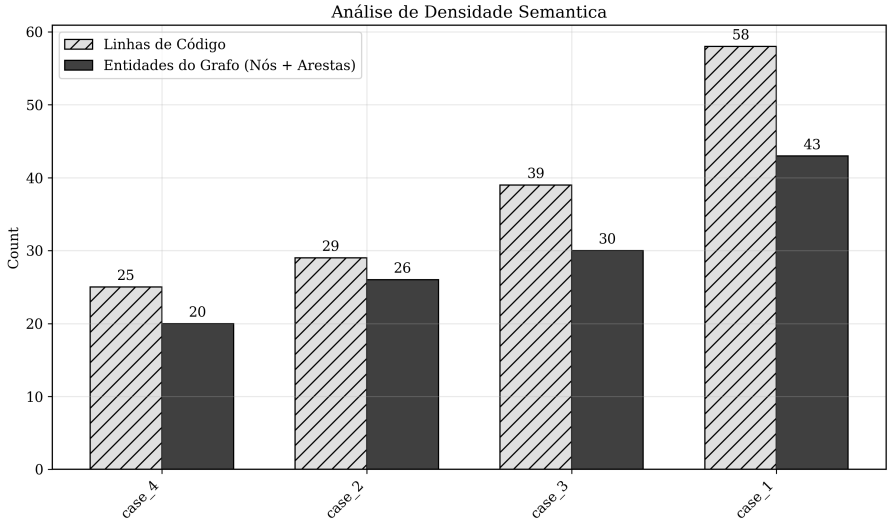

Figura 3. Densidade Semântica: Comparação entre linhas de código e elementos do grafo gerado.

5.3. Análise Subjetiva de Escalabilidade

Ao invés de uma medição empírica de tempo, propomos uma análise subjetiva baseada em um cenário de expansão de IaC para ilustrar o impacto da solução na carga cognitiva operacional.

Considere uma situação em que um engenheiro DevOps é responsável pela manutenção de um ambiente de microsserviços fragmentado em 20 repositórios distintos, onde cada equipe define seu próprio arquivo *docker-compose*. Suponha que uma nova aplicação precise ser implantada expondo a porta 8080 no *host*. Não existe um registro centralizado das portas em uso; a informação está dispersa em 20 arquivos estáticos.

Neste cenário, para garantir que não haverá conflito de porta (que resultaria em um erro no serviço em produção), o operador precisaria abrir e inspecionar visualmente cada um dos arquivos existentes, cruzando mentalmente as informações. A complexidade dessa verificação manual cresce de uma forma não linear: a cada serviço adicionado, o esforço cognitivo aumenta e a probabilidade de erro humano (esquecer uma porta já vista) torna-se crítica.

Ao transformar a infraestrutura em um Grafo de Conhecimento, o custo de verificação deixa de ser proporcional ao número de arquivos e se torna uma consulta em tempo constante. O agente MCP não inspeciona cada arquivo sequencialmente, ele apenas questiona o grafo se o nó `:PortMapping hostPort: 8080` já possui uma aresta de entrada, retornando o conflito instantaneamente, independentemente se o ambiente possui 10 ou 1000 serviços.

5.4. Estudos de Caso: Raciocínio Semântico

Para validar a profundidade do raciocínio habilitado pelo MCP, o sistema foi submetido a quatro cenários de auditoria que exigem compreensão contextual da infraestrutura.

Cenário A: Análise de Impacto em Cadeia Em arquiteturas distribuídas, falhas em serviços de infraestrutura propagam-se silenciosamente. Este experimento modelou

uma pilha de e-commerce com quatro camadas de dependência: `ecommerce-db` → `inventory-api` → `session-cache` → `storefront-ui`. Ao simular uma falha no nó (`ecommerce-db`), a ferramenta de análise de dependência identificou corretamente que o serviço `storefront-ui` seria impactado, mesmo não possuindo uma conexão direta com o banco de dados no arquivo de definição. A análise léxica tradicional, falharia em traçar esse caminho de quatro saltos, invisível em uma leitura superficial ou sem cuidado.

Cenário B: Integridade de Volumes O isolamento de containers não protege o sistema de arquivos do *host* contra concorrência de escrita. Neste cenário, dois projetos distintos foram configurados erroneamente para montar o mesmo diretório local para persistência de dados. O agente MCP detectou a colisão do *HostPath*, alertando preventivamente sobre uma condição crítica de corrupção de dados que o *runtime* do Docker só manifestaria durante a operação simultânea de escrita.

Cenário C: Detecção de Movimentação Lateral A segurança de redes em ambientes conteinerizados depende de uma segmentação rigorosa entre zonas. Neste cenário, foram definidas duas redes distintas: uma zona pública (`public_internet`) e uma zona segura (`secure_vault`). Um serviço intermediário legado, denominado `legacy-auth-service`, foi conectado simultaneamente a ambas as redes com o objetivo de viabilizar um processo de migração. Ao analisar a segurança do banco de dados `vault-database`, o Agente MCP identificou um caminho transitivo não autorizado. Observou-se que um atacante, após comprometer o `load-balancer` na zona pública, poderia alcançar a zona segura utilizando o `legacy-auth-service` como ponto de pivô. Essa configuração caracteriza uma violação da política de segmentação de rede, que não era evidente na análise isolada dos serviços, mas revelada pela modelagem em grafo.

Cenário D: Conflito de Portas Este cenário replicou um problema comum em ambientes de teste paralelos. Foram instanciados dois serviços de banco de dados em redes virtuais distintas. Embora estivessem logicamente isolados, ambos foram configurados para escutar na mesma porta do *host*. Ferramentas tradicionais validam o arquivo individualmente e não detectam o erro. O agente, ao consultar o grafo global, identificou que dois nós de serviços distintos convergiam para o mesmo nó de recurso `PortMapping`, alertando sobre a impossibilidade física da implantação simultânea.

A Tabela 1 sumariza a eficácia da abordagem proposta frente aos métodos tradicionais. Observa-se que ferramentas padrão se limitam ao escopo do arquivo individual, resultando em falsos-negativos para problemas que dependem do estado global da infraestrutura.

Tabela 1. Comparativo de Capacidades de Detecção entre Abordagens

Tipo de Falha	Docker CLI	Linter Estático	Grafo + MCP (Proposto)
Erro de Sintaxe YAML	Sim	Sim	Sim
Conflito de Porta (Local)	Sim	Sim	Sim
Conflito de Porta (Global/Host)	Não	Não	**Sim**
Dependência Transitiva ($N > 1$)	Não	Não	**Sim**
Violação de Segmentação de Rede	Não	Não	**Sim**
Conflito de Volume (Host Path)	Não	Parcial	**Sim**

6. Discussão

Este trabalho apresenta uma nova estrutura para o gerenciamento de infraestrutura conteinerizada, integrando Grafos de Conhecimento baseados em ontologias com o MCP. Ao transformar artefatos estáticos de IaC em um grafo semântico, a solução preenche a lacuna existente entre arquivos de configuração brutos e o raciocínio dinâmico necessário para a orquestração de sistemas complexos.

Os resultados evidenciam que a abordagem proposta supera a análise estática tradicional. Enquanto *linters* de YAML se limitam à validação sintática local, o Grafo de Conhecimento permite auditorias globais e semânticas, permitindo a detecção de problemas que exigem compreensão contextual, como dependências transitivas e validações de segmentação de rede. Além disso, a análise subjetiva demonstrou que a centralização das informações no grafo reduz drasticamente a carga cognitiva operacional, transformando a verificação manual de múltiplos arquivos em consultas de tempo constante.

A integração com LLMs via MCP provou ser um diferencial arquitetural robusto. Ao desacoplar o modelo de linguagem da base de dados através de ferramentas determinísticas, a estrutura minimizou significativamente o risco de alucinações e erros de sintaxe (comuns na geração direta de queries como Cypher e SQL). Isso confirma a hipótese de que o enriquecimento semântico da IaC, aliado a um protocolo de contexto padronizado, permite que a inteligência artificial atue como um orquestrador de raciocínio confiável e não apenas como um gerador de código.

No entanto, é importante ressaltar as limitações da abordagem atual. A precisão do Grafo de Conhecimento depende estritamente da declaração e qualidade dos arquivos de infraestrutura. Configurações injetadas via scripts de inicialização ou variáveis de ambiente definidas apenas no tempo de execução (sem valores padrão no arquivo) escapam ao escopo da análise.

7. Próximos Passos

Para aprimorar ainda mais as capacidades dessa estrutura e mitigar as limitações identificadas, o desenvolvimento futuro se concentrará na evolução do sistema de um modelo passivo para um modelo ativo de remediação. O objetivo é permitir que o agente execute tarefas de correção de forma autônoma, como o isolamento ou desligamento preventivo de contêineres com comportamento anômalo, fundamentado no raciocínio lógico extraído do grafo do conhecimento.

Além disso, planeja-se estender a ontologia para suportar dados históricos e o estado em tempo real. Isso permitirá o monitoramento dinâmico da capacidade de recursos e a comparação do estado atual versus o estado desejado definido nas políticas da infraestrutura. Essa evolução visa eliminar a dependência exclusiva de arquivos estáticos, oferecendo uma supervisão contínua da integridade e das dependências da infraestrutura, sem a necessidade de intervenção humana na análise de configurações de baixo nível.

Agradecimentos: O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES). Bolsa PROEX 88887.177877/2025-00 - Código de Financiamento 001, e com o apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPQ) - Brasil. Bolsa PQ, processo nº 308075/2025-0 e processo nº 315427/2023-0.

Referências

Boukadi, K. et al. (2020). Container description ontology for caas. *International Journal of Web and Grid Services*, 16(4):341. Accessed: 2025-11-18.

Claise, B. (2025). Knowledge graph as preparation for llms, the logical next step in networking data modeling! – benoît claise. Online: https://www.claise.be/knowledge-graph-as-preparation-for-llms-the-logical-next-step-in-networking-data-modeling/. Accessed: 2026-01-26.

Docker (2025). Docker. Online: https://www.docker.com/. Accessed: 2025-11-26.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220.

IBM and Stryker, C. (2023). What are large language models (llms)? Online: https://www.ibm.com/think/topics/large-language-models.

Kerner, S. M. (2025). Rise of containers: How enterprises are adapting to cloud-native demands. Online: https://www.itprotoday.com/cloud-computing/cloud-infrastructure-reaches-turning-point-as-container-adoption-becomes-universal. Accessed: 2026-01-19.

Kyriakopoulos, K. G., Chilton, P., and Parish, D. (2015). Flowstats: An ontology based network management tool. In *Proceedings of the International Conference on Computing Technology and Information Management (ICCTIM)*, pages 13–18. Accessed: 2025-08-08.

Martinez-Casanueva, I. D. (2025). Knowledge graphs for yang-based network management. Online: https://datatracker.ietf.org/doc/draft-marcas-nmop-knowledge-graph-yang/00/. Accessed: 2025-11-26.

Nutanix (2025). Nutanix enterprise cloud index. Online: https://www.nutanix.com/enterprise-cloud-index. Accessed: 2026-01-28.

Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., and Wu, X. (2024). Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599.

Red Hat (2025). What are large language models? Online: https://www.redhat.com/en/topics/ai/what-are-large-language-models.

Shool, S., Adimi, S., Amleshi, R. S., Bitaraf, E., Golpira, R., and Tara, M. (2025). A systematic review of large language model (llm) evaluations in clinical medicine. *BMC Medical Informatics and Decision Making*, 25.

Simić, A. and Palma, D. (2024). Using knowledge graphs to automate network compliance of containerized services. In *2024 20th International Conference on Network and Service Management (CNSM)*, pages 1–5.

Singhal, A. (2012). Introducing the knowledge graph: Things, not strings. Online: https://blog.google/products-and-platforms/products/search/introducing-knowledge-graph-things-not/. Accessed: 2026-01-26.

Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1-2):161–197.

Vasileiou, Z., Kumara, I., Meditskos, G., Tokmakov, K., Radolović, D., Cruz, J., Nitto, E., Tamburri, D., Heuvel, W.-J., and Vrochidis, S. (2025). A knowledge-based approach for guided development of infrastructure as code. *Software and Systems Modeling*, pages 1–34.

Wang, W. et al. (2026). PTFusion: LLM-driven context-aware knowledge fusion for web penetration testing. *Information Fusion*, 127:103731.

Wu, S., Irsoy, O., Lu, S., Dabravolski, V., Dredze, M., Gehrmann, S., Kambadur, P., Rosenberg, D. S., and Mann, G. (2023). Bloomberggpt: A large language model for finance. *ArXiv*, abs/2303.17564.

Yao, Y., Duan, J., Xu, K., Cai, Y., Sun, Z., and Zhang, Y. (2024). A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4:100211.

Zhou, H., Hu, C., Yuan, Y., Cui, Y., Jin, Y., Chen, C., Wu, H., Yuan, D., Jiang, L., Wu, D., Liu, X., Zhang, J., Wang, X., and Liu, J. (2025). Large language model (llm) for telecommunications: A comprehensive survey on principles, key techniques, and opportunities. *IEEE Communications Surveys & Tutorials*, 27(3):1955–2005.

Zhou, J. et al. (2020). DockerKG: A knowledge graph of docker artifacts. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 367–372. Accessed: 2025-11-25.