AgentK: detecção inteligente de erros em arquivos de configuração do Kubernetes, uma implementação do Model Context Protocol

AgentK: intelligent error detection in Kubernetes configuration files, an implementation of the Template Contex Protocol

Engenharia de Computação

Vinicius Gabriel da Silva Olimpio *(viniolimpio3@gmail.com)*
Graduado em Engenharia de Computação pela Faculdade Engenheiro Salvador Arena (FESA).

David Santos Pinheiro das Neves *(davidspdn@gmail.com)*
Graduado em Engenharia de Computação pela Faculdade Engenheiro Salvador Arena (FESA).

Cleber dos Santos Santana *(081210015@faculdade.cefsa.edu.br)*
Graduado em Engenharia de Computação pela Faculdade Engenheiro Salvador Arena (FESA).

Wilson Pereira da Silva *(081210039@faculdade.cefsa.edu.br)*
Graduado em Engenharia de Computação pela Faculdade Engenheiro Salvador Arena (FESA).

Fábio Henrique Cabrini *(pro10964@cefsa.edu.br)*
Doutor em Engenharia Elétrica pela Universidade de São Paulo (USP) e professor da Faculdade Engenheiro Salvador Arena (FESA) e da FIAP.

FTT Journal of Engineering and Business
• SÃO BERNARDO DO CAMPO, SP JUN. 2026
• ISSN 2525-8729
Submissão: 19 dez. 2025 Aceitação: 22 maio 2026
Sistema de avaliação: às cegas dupla (double blind review)
FACULDADE ENGENHEIRO SALVADOR ARENA, p. 109 – p. 125

FTT JOURNAL of Engineering and Business

FACULDADE ENGENHEIRO SALVADOR ARENA

Resumo

O crescimento do uso de contêineres e sua orquestração com Kubernetes tem trazido benefícios de escalabilidade e padronização, mas também desafios relacionados a falhas de configuração, responsáveis por grande parte dos incidentes de segurança na plataforma. Nesse cenário, este trabalho apresenta o AgentK, um sistema baseado em *Large Language Models* (LLM) e no *Model Context Protocol* (MCP), desenvolvido para a detecção inteligente de erros em arquivos YAML (Ain't *Markup Language*) do Kubernetes. A pesquisa foi conduzida em ambiente controlado com Minikube e infraestrutura do provedor de nuvem *Amazon Web Service* (AWS), utilizando arquivos com falhas intencionais. Foram comparados diferentes modelos quanto a custo, tempo de resposta e capacidade de detecção. Os resultados mostraram que o o4-mini obteve 100% de acurácia, mas apresentou limitações de integração; já o gpt-4.1, embora tenha identificado 66,7% das falhas, foi escolhido por oferecer melhor equilíbrio no quesito custo, que representa cerca de 28,5% menos que o o4-mini, desempenho e compatibilidade arquitetural. A arquitetura final permitiu não apenas detectar, mas também corrigir falhas automaticamente, evidenciando o potencial do AgentK para reforçar a segurança e a robustez de ambientes Kubernetes, especialmente em cenários críticos como cidades inteligentes.

Palavras-chave: Kubernetes. Modelo de Linguagem de Grande Escala. Cidades inteligentes. Modelo de Protocolo de Contexto

Abstract

The increasing adoption of containers and their orchestration through Kubernetes has brought scalability and standardization benefits, but also challenges related to configuration errors, which account for a substantial portion of security incidents on the platform. In this context, this paper presents AgentK, a system based on Large Language Models (LLM) and the Model Context Protocol (MCP), developed for the intelligent detection of configuration errors in Kubernetes (YAML *Ain't Markup Language*) files. The research was conducted in a controlled environment using Minikube and cloud provider Amazon Web Service (AWS) infrastructure, with validation performed through files containing intentional misconfigurations. Several language models were compared in terms of cost, response time, and detection capability. Results showed that the o4-mini model reached 100% accuracy but presented integration limitations; meanwhile, gpt-4.1, although identifying 66.7% of the issues, was selected for offering a better balance between cost, approximately 28.5% lower than o4-mini, performance, and architectural compatibility. The final architecture enabled not only error detection but also automated correction, demonstrating AgentK's potential to strengthen security and robustness in Kubernetes environments, especially in critical scenarios such as smart cities.

Keywords: Kubernetes. Large Language Model. Smart Cities. Model Context Protocol.

Introdução

O Kubernetes consolidou-se como plataforma essencial para orquestração de contêineres em produção, permitindo implantação, escalonamento e monitoramento automatizados de aplicações (Muralidharan, Song e Ko, 2019). A **Cloud Native Computing Foundation** (2024) indica que 93% das organizações já utilizam a plataforma, embora enfrentem desafios relacionados à complexidade técnica (35%) e à segurança (37%).

O State of Kubernetes Security Report (2022) revela que 53% dos incidentes de segurança originam-se de erros nos arquivos YAML, responsáveis por definir permissões e políticas dos componentes do cluster. Esse problema torna-se crítico em cidades inteligentes, onde múltiplos domínios (mobilidade, energia, saúde) dependem de aplicações distribuídas processando dados em tempo real (Gracias *et al.*, 2023). Qualquer falha computacional compromete o sistema urbano como um todo, reforçando a necessidade de infraestruturas confiáveis.

Os contêineres oferecem portabilidade, enquanto o Kubernetes viabiliza seu gerenciamento em larga escala (Matthias e Kane, 2016). Contudo, a ausência de mecanismos automatizados para a validação de configurações amplifica a exposição a vulnerabilidades, exemplificada pelo incidente na Tesla em 2018, quando uma instância Kubernetes sem autenticação foi comprometida (BBC News, 2018).

Nesse contexto, os Large Language Models (LLM) emergem como alternativa promissora, dada sua capacidade de interpretar estruturas textuais complexas, como arquivos YAML, de forma contextual (Minaee *et al*., 2025). Este trabalho pressupõe que a integração de LLM a sistemas automatizados pode aprimorar a identificação e a mitigação de erros de configuração em Kubernetes.

O objetivo geral é desenvolver um sistema automatizado capaz de analisar arquivos YAML, verificar conformidade com boas práticas e implementar correções. Os objetivos específicos são: (i) desenvolver um agente inteligente para identificar erros com precisão mínima de 75%; (ii) implementar mecanismos automatizados de correção e validação estrutural; e (iii) projetar uma interface conversacional que permita interação eficiente entre usuário e sistema.

Referencial teórico

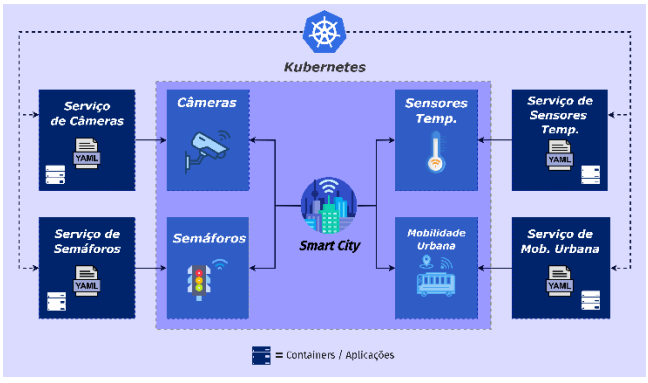
Conteinerização

De acordo com Matthias e Kane (2016), os contêineres não devem ser confundidos com *virtual machine* (VM), pois funcionam como uma camada leve de abstração sobre o sistema operacional, utilizando poucos recursos e iniciando rapidamente. Ao contrário da VM, os contêineres são efêmeros, podendo ser criados e destruídos com agilidade, o que os torna ideais para ambientes que exigem escalabilidade e eficiência no uso do *hardware*.

Kubernetes

Segundo Lukša (2018), o Kubernetes fornece abstração sobre a infraestrutura de hardware, permitindo o gerenciamento automatizado de aplicações em contêineres com foco em escalabilidade e resiliência. A plataforma utiliza objetos, entidades persistentes que representam o estado desejado do cluster, descritos em arquivos YAML, uma linguagem de serialização com elevada legibilidade, também denominados *manifests* (Kubernetes, 2025; YAML, 2021).

Figura 1: Kubernetes como orquestrador dos serviços de cidades inteligentes



Fonte: elaboração dos autores (2025).

Minikube

O Minikube é uma ferramenta de código aberto que permite criar rapidamente um *cluster* Kubernetes local em sistemas operacionais Linux, macOS e Windows (Minikube, 2025). Seu principal objetivo é auxiliar desenvolvedores de aplicações e novos usuários do Kubernetes, oferecendo uma maneira simples de executar um *cluster* de nó único em uma máquina local para fins de desenvolvimento e testes.

Large Language Model

Os modelos de LLM representam um avanço significativo em processamento de linguagem natural, possibilitando de tarefas simples como classificação a aplicações complexas com múltiplas etapas de raciocínio (Minaee *et al*., 2025). Baseados na arquitetura Transformer, esses modelos utilizam um mecanismo de atenção capaz de analisar grandes quantidades de texto em paralelo e identificar relações entre palavras, tornando-os altamente eficientes para a compreensão de linguagem e geração de texto (Xu *et al*., 2024).

Agentes de inteligência artificial

Os agentes de IA representam uma geração de sistemas inteligentes construídos sobre modelos LLM, transcendendo a simples geração de texto (Wang *et al*., 2024; Shao *et al*., 2025). São sistemas orientados a objetivos, capazes de planejar autonomamente, executar ações através de ferramentas e APIs e monitorar múltiplas etapas de uma tarefa. A implementação desses agentes é facilitada por protocolos como o MCP, que padroniza a comunicação entre LLM e sistemas externos (Hou *et al*., 2025; Ray, 2025).

Model Context Protocol

O MCP é um protocolo aberto que facilita a comunicação entre LLM e sistemas externos de forma segura, extensível e padronizada (Ray, 2025). Opera sobre JSON-RPC 2.0, permitindo comunicação assíncrona e bidirecional entre cliente e servidor (Hou *et al*., 2025). Sua estrutura define três componentes: o *host* (aplicação cliente), o cliente MCP (gerencia conexões) e o servidor MCP (expõe ferramentas e recursos). Essa separação de responsabilidades promove modularidade e facilita a manutenção, consolidando-se como base promissora para aplicações seguras com LLM, já adotada por empresas como OpenAI, Anthropic e Cloudflare.

Ferramentas de validação de configurações Kubernetes

Diversas ferramentas foram desenvolvidas para auxiliar na validação de arquivos de configuração Kubernetes. O Kubeval e seu sucessor, Kubeconform, realizam validação estática de schema, verificando se os arquivos YAML estão estruturalmente conformes com a especificação da API do Kubernetes (Kubeconform, 2023). O Datree amplia essa abordagem, permitindo a definição de políticas customizadas de boas práticas a serem verificadas automaticamente em pipelines de integração contínua (Datree, 2023). O Conftest, baseado no motor de políticas OPA (Open Policy Agent), possibilita validações mais expressivas por meio de regras escritas na linguagem Rego (Conftest, 2023). O kube-score, por sua vez, realiza análise estática atribuindo pontuações aos recursos com base em práticas recomendadas de segurança e confiabilidade (Kube-score, 2023).

Embora eficazes para verificações estruturais e conformidade com políticas predefinidas, essas ferramentas operam com base em regras estáticas, sem capacidade de raciocínio contextual. O AgentK diferencia-se ao empregar LLMs para interpretar semanticamente o conteúdo dos arquivos YAML, identificando falhas que demandam compreensão contextual e sugerindo correções de forma conversacional e adaptativa.

Metodologia

Ambiente experimental

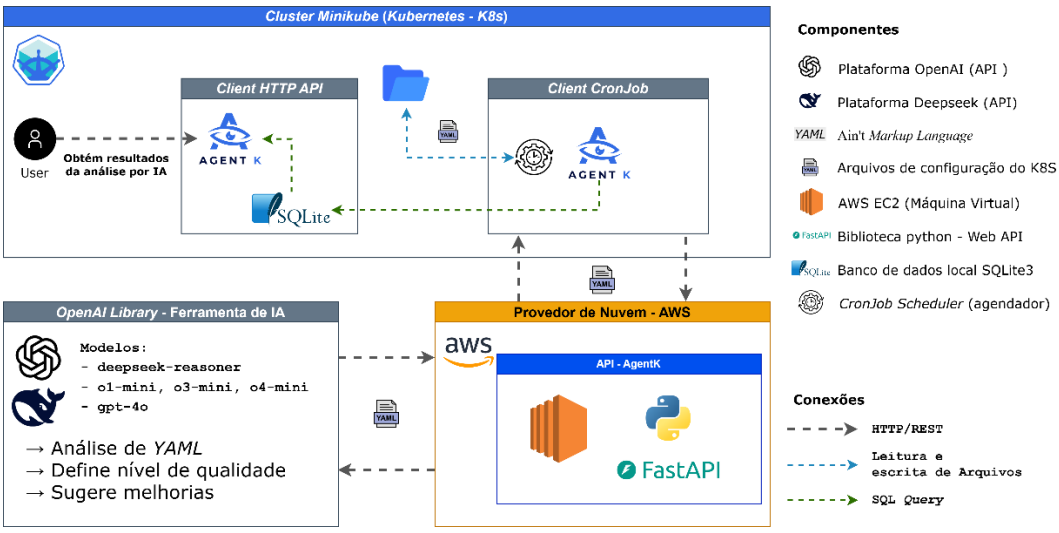
O AgentK foi desenvolvido em ambiente de laboratório utilizando Minikube, posteriormente hospedado na infraestrutura AWS. Conforme Kruger (2023), ambientes controlados permitem estabelecer condições próximas do ambiente real, identificando fenômenos que poderiam passar despercebidos. Foi utilizada uma máquina virtual com *Ubuntu Server* 24.04 LTS, configurada com 60 GB de armazenamento, 2 vCPUs e 4 GB de RAM, especificações consideradas mínimas para execução adequada. As versões utilizadas foram: Docker 27.3.1, kubectl 1.35.0 e Minikube 1.35. Detalhes de configuração e *deployment* estão disponíveis no repositório oficial do projeto (Olimpio *et al.*, 2025a).

Com base na documentação oficial do Kubernetes, foram definidas diretrizes de boas práticas que o AgentK utilizou como referência para avaliação. Entre elas incluíram-se: a definição explícita de

imagens com *tags*, o uso de *liveness probe*, a limitação de recursos, boas práticas de segurança e separação de configuração por *Secrets* e *ConfigMaps*. Essas recomendações foram incorporadas em um *prompt* de modelo de linguagem, disponível no repositório público do projeto, juntamente com um *schema* em formato JSON utilizado para padronizar as respostas do modelo e garantir consistência na análise (OLIMPIO *et al.*, 2025a).

A figura 2 ilustra a arquitetura desenhada para a primeira versão do AgentK. O fluxo iniciava-se no *client cronJob*, executado a cada cinco minutos, que extraía arquivos de extensão.*yaml* de uma pasta configurada como variável de ambiente no contexto do *cluster* Minikube. O conteúdo desses arquivos de configuração era encaminhado à API do AgentK, que realizava a interface com os grandes modelos de LLM, conforme *prompt* e *schema* de resposta definidos. As avaliações geradas pelo LLM (*score*, recomendações e relatório de falhas) eram mantidas em um banco de dados local SQLite e disponibilizadas por meio de uma API para visualização do usuário.

Figura 2 – Arquitetura inicial do AgentK: fluxo baseado em CronJob.



Fonte: elaboração dos autores (2025)

Durante o desenvolvimento do protótipo, foi identificada a necessidade de uma abordagem robusta e interativa para o sistema. A arquitetura inicial, baseada em *CronJob*, apresentou limitações de usabilidade e flexibilidade para interação em tempo real com o conteúdo dos arquivos de configuração do Kubernetes. Consequentemente, o sistema foi reestruturado, adotando o MCP e uma interface *web* conversacional; o servidor MCP expõe ferramentas indicadas no quadro 1, possibilitando um padrão para a comunicação entre o usuário e as interações com o modelo de linguagem.

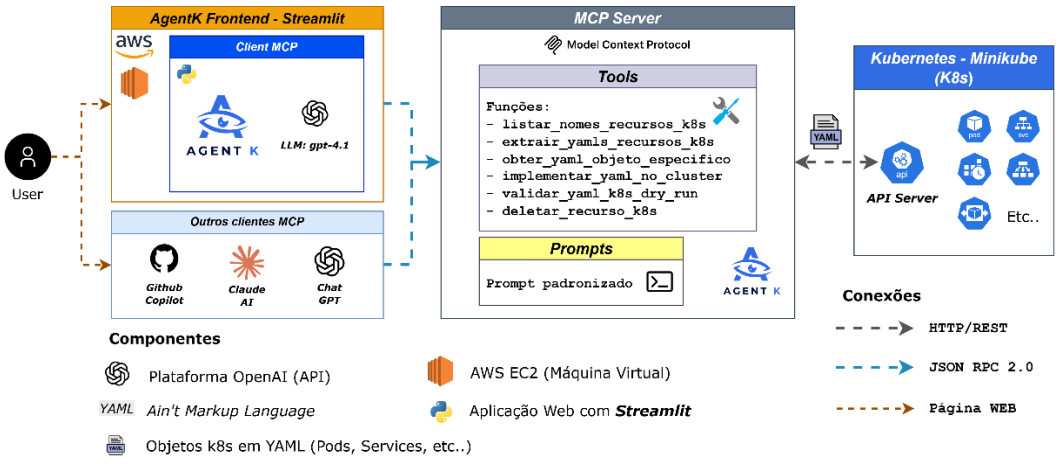
Quadro 1 – Descrição das funções desenvolvidas no servidor MCP.

Nome da função (Tool)	Descrição
listar_nomes_recursos_k8s	Lista recursos do *cluster* organizados por tipo (*Pods*, *Services*, *Deployments*, *Cronjobs*, etc.)
extrair_yamls_recursos_k8s	Extrai configurações YAML completas de recursos existentes
obter_yaml_objeto_especifico	Obtém configurações específicas por nome e *namespace*
implementar_yaml_no_cluster	Aplica recursos no *cluster*, com validação de estrutura do yaml
validar_yaml_k8s_dry_run	Valida estrutura YAML de um arquivo, sem a efetivação
deletar_recurso_k8s	Remove recursos específicos do *cluster*

Fonte: elaboração dos autores (2025).

A nova arquitetura está representada na figura 3. Desenvolveu-se um cliente *web* (*frontend*) utilizando o *Streamlit* como *framework Python* de interface gráfica, hospedado em uma instância AWS EC2 (*Elastic Cloud Computing*). O *frontend* implementa uma interface conversacional que permite ao usuário interagir em linguagem natural com o agente de IA que, por sua vez, pode executar ações diretamente no *cluster* Kubernetes através de *chat*. Integrou-se o modelo gpt-4.1 como base do agente conversacional, que atua como especialista em configurações YAML Kubernetes, e um agente de boas práticas. O cliente estabelece comunicação com o servidor MCP através do protocolo JSON RPC 2.0, o padrão para o protocolo MCP. O servidor MCP, por sua vez, foi desenvolvido utilizando a biblioteca *python FastMCP*, atuando diretamente com as bibliotecas oficiais do Kubernetes *Python Client*. Tal servidor tem a proposta de disponibilizar funções específicas, que são executadas pela indicação do LLM.

Figura 3 – Arquitetura final do AgentK.



Fonte: elaboração dos autores (2025).

Seleção de arquivos YAML

Para validar o funcionamento da nova arquitetura, foram realizados testes independentes utilizando dez arquivos YAML obtidos do repositório oficial de exemplos do Kubernetes no Github (Kubernetes, [s.d.-b]), selecionados com base em dois critérios principais: compatibilidade com o ambiente experimental e diversidade de recursos. O critério de compatibilidade exigiu que cada arquivo representasse uma configuração executável no ambiente Minikube disponível, respeitando as limitações de hardware da máquina virtual utilizada, ou seja, os pods deveriam atingir o estado *Running* com razão 1/1 após a aplicação. Buscou-se também contemplar diferentes tipos de recursos Kubernetes, como *Deployments* de aplicações variadas, ainda que sem uma estratificação formal por tipo de recurso. Os arquivos selecionados representam configurações reais da plataforma e serviram como base para a inserção intencional das falhas descritas na próxima seção.

Tipos de falhas inseridas

Em cada um dos dez arquivos YAML selecionados foram inseridas intencionalmente três categorias de falhas, escolhidas por sua recorrência em ambientes Kubernetes reais e por permitirem a avaliação de capacidades distintas do AgentK.

A primeira categoria consiste em credenciais expostas, caracterizada pela presença de senhas, chaves ou outros dados sensíveis definidos diretamente no corpo do arquivo YAML, sem uso de recursos nativos do Kubernetes como *Secrets* ou *ConfigMaps*. Essa classe de falha representa um dos principais vetores de incidentes de segurança na plataforma (State of Kubernetes security report, 2022).

A segunda categoria refere-se à ausência de especificação de versão da imagem, situação em que o campo *image* é definido sem uma *tag* explícita ou com o uso da *tag latest*, tornando o ambiente suscetível a variações de comportamento entre implantações e dificultando a rastreabilidade das versões em produção.

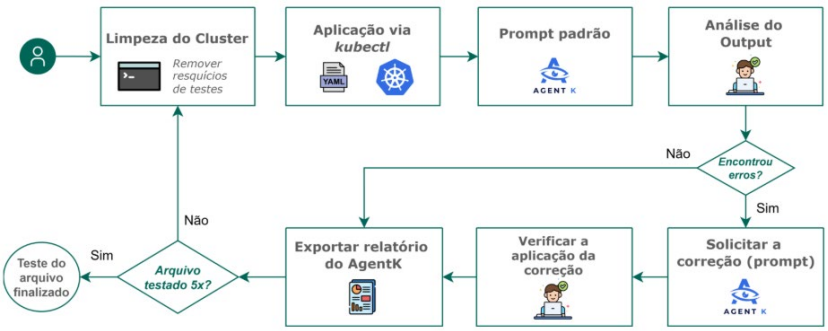
A terceira categoria engloba erros semânticos, que incluem inconsistências de *label mismatch* entre recursos relacionados (como *Deployment* e *Service*), referências a imagens inexistentes, comandos inválidos e caminhos incorretos em campos de configuração. Esse tipo de falha é

particularmente relevante por não ser detectável por validadores estáticos de schema, exigindo compreensão contextual do conteúdo do arquivo.

Protocolo de teste

Cada arquivo foi submetido a cinco rodadas de testes, totalizando 50 execuções do protocolo experimental, conforme o fluxo representado na figura 4. Cada rodada seguiu a sequência padronizada: limpeza do cluster via *kubectl delete all --all -n default*; aplicação do arquivo com falhas via *kubectl apply*; envio de prompt padrão ao AgentK solicitando análise e correção das falhas e verificação do estado final dos *pods* via *kubectl get pods*, considerando sucesso apenas os cenários em que todos os *pods* atingiram o estado *Running* com razão 1/1. Os registros de cada sessão estão disponíveis no repositório público do projeto (Olimpio et al., 2025b).

Figura 4: Passo a passo do teste de detecção de erro do AgentK.



Fonte: elaboração dos autores (2025).

Limitações

O presente estudo apresenta limitações que devem ser consideradas na interpretação dos resultados. Primeiramente, foram avaliados apenas três tipos de falhas: credenciais expostas, ausência de tag de imagem e erros semânticos. Trata-se de delimitação adotada por razões práticas de escopo, que não representa a totalidade de falhas recorrentes em ambientes Kubernetes reais, como ausência de NetworkPolicies, uso de privilégios elevados (runAsRoot) ou configurações inadequadas de RBAC. Em segundo lugar, os testes foram conduzidos em cluster Minikube de nó único com recursos de hardware restritos, o que pode não refletir o comportamento do sistema em infraestruturas de produção de maior escala. Por fim, os dez arquivos YAML utilizados, embora obtidos do repositório oficial do Kubernetes, não cobrem a totalidade de tipos de recursos da plataforma, limitando a generalização dos resultados. A

expansão do conjunto de falhas, ambientes e tipos de recursos constitui direção prioritária para trabalhos futuros.

Resultados e discussão

O quadro 2 apresenta os resultados preliminares obtidos durante essa fase de testes. Foram avaliados 8 modelos diferentes: gpt-4o, o4-mini, o3-mini, o1-mini, gpt-4.1, o1, gpt-3.5-turbo e deepseek-reasoner; alguns modelos avaliados podem ter sido descontinuados posteriormente, porém estavam disponíveis no período em que foram feitos os testes e foram considerados relevantes para comparação. Cada modelo foi avaliado de maneira independente, registrando-se tempo de resposta, quantidade de *input* e *output tokens*, custo em dólares por 100 requisições e a quantidade de problemas identificados corretamente dentre as três presentes no arquivo de teste. Todas as execuções utilizaram o mesmo *prompt*, o mesmo *schema* JSON. Os testes foram realizados em maio de 2025 e os custos foram calculados, sendo possível ver os preços das API na data dos experimentos.

Quadro 2 - Resultados preliminares obtidos com a utilização do software AgentK.

Modelo	Input tokens	Output tokens	Tempo de resposta	Custo / 10 requisições (US$)	Credenciais expostas	Versão de imagem	Erro de sintaxe
gpt-4o	1198	1056	1 min 32 s	0,14	Detectado	Não detectado	Detectado
o4-mini	1197	3458	1 min 56 s	0,17	Detectado	Detectado	Detectado
o3-mini	1197	1338	0 min 13 s	0,07	Detectado	Não detectado	Detectado
gpt-4.1	1198	1180	0 min 55 s	0,12	Detectado	Não detectado	Detectado
o1	1197	6187	0 min 46 s	3,89	Detectado	Detectado	Detectado
gpt-3.5-turbo	1202	749	0 min 12 s	0,02	Detectado	Não detectado	Detectado
deepseek-reasoner	1326	8082	4 min 18 s	0,18	Detectado	Não detectado	Detectado

Fonte: elaboração dos autores (2025)

Com base nesses dados técnicos e operacionais, foi selecionado o modelo gpt-4.1 para a implementação final. Embora os modelos o3-mini e gpt-3.5-turbo tenham apresentado assertividade equivalente na detecção de erros, com custo e tempo de resposta menores, ambos foram descartados por razões distintas: o gpt-3.5-turbo foi oficialmente descontinuado pela OpenAI durante o período de desenvolvimento do projeto, tornando-o inviável para uso em produção; já o o3-mini, apesar de disponível, não oferecia compatibilidade nativa com a API *completions* já

implementada na arquitetura do AgentK, o que exigiria refatoração significativa do código de integração sem ganho proporcional em desempenho. O modelo o4-mini, por sua vez, embora tenha alcançado 100% de acurácia na detecção, também apresentou limitações de integração com a API *completions*, além de custo aproximadamente 28,5% superior ao gpt-4.1. Dessa forma, o gpt-4.1 mostrou-se a escolha mais equilibrada nos quesitos custo, desempenho e compatibilidade arquitetural, enquanto o uso de o1 permaneceu inviável em escala devido ao custo elevado.

Os experimentos realizados com o AgentK permitiram avaliar sua capacidade de detectar falhas em arquivos de configuração do Kubernetes e o desempenho dos modelos de linguagem empregados na análise. Para a validação do sistema, foram utilizados dez arquivos YAML obtidos da documentação oficial do Kubernetes, que representam configurações reais de recursos da plataforma. Esses arquivos serviram como base para os testes de detecção e correção de erros, sendo inseridos propositalmente três tipos de falhas recorrentes (credenciais expostas, ausência de especificação da versão da imagem e erro de sintaxe no campo *readinessProbe)* a fim de garantir uniformidade e comparabilidade nos resultados. A tabela 1 mostra o percentual de sucesso na detecção dos erros.

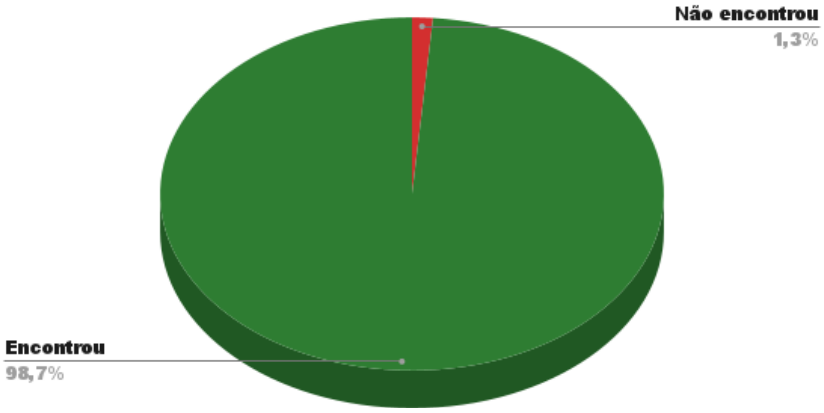
Tabela 1 – Resultados de detecção de erros obtidos com a utilização do AgentK.

Arquivo	Detecção de erro de credenciais expostas (%)	Detecção de erro semântico (%)	Detecção de erro de versão de imagem (%)
1-orion.yaml	100	100	100
2-frontend.yaml	100	100	100
3-mysql.yaml	100	60	100
4-vllm.yaml	100	100	100
5-nginx.yaml	100	100	100
6-selenium.yaml	100	100	100
7-elasticsearch.yaml	100	100	100
8-newrelic.yaml	100	100	100
9-storm.yaml	100	100	100
10-mongodb.yaml	100	100	100
Acurácia	100	96	100

Fonte: elaboração dos autores (2025)

No processo, foi utilizado *prompt* padrão para as solicitações ao agente na detecção dos erros credenciais expostas, erro semântico e versão da imagem. O teste foi repetido 5 vezes para cada arquivo, totalizando 150 testes, nos quais o AgentK encontrou o erro corretamente 148 vezes, como resultado global mostrado no gráfico 1.

Gráfico 1: Precisão na detecção de erros do AgentK.



Fonte: elaboração dos autores (2025)

Para cada um desses arquivos, foi realizado o teste de correção automática. Neste teste, após a detecção do erro, o usuário solicita via *chat* ao AgentK sugerir e implantar a correção necessária para corrigir o problema. A tabela 2 mostra os resultados para cada arquivo.

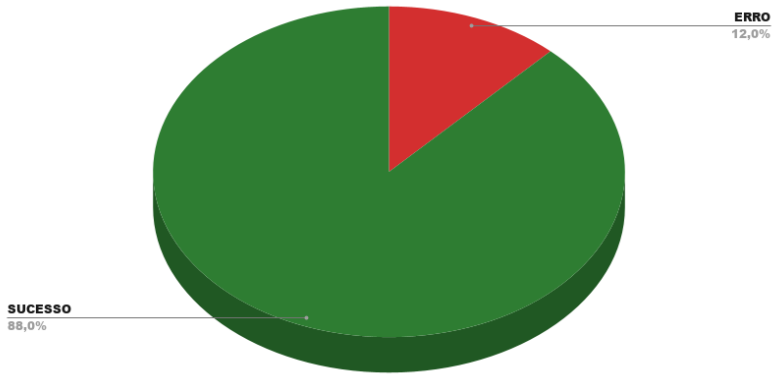
Tabela 2 - Resultados de sucesso na aplicação de correção automática com a utilização do AgentK.

Arquivo	Sucesso na correção automática (%)
1-orion.yaml	100
2-frontend.yaml	100
3-mysql.yaml	60
4-vllm.yaml	80
5-nginx.yaml	80
6-selenium.yaml	80
7-elasticsearch.yaml	100
8-newrelic.yaml	100
9-storm.yaml	100
10-mongodb.yaml	80
Acurácia	88

Fonte: elaboração dos autores (2025).

Novamente, o teste foi repetido cinco vezes para cada arquivo. Os resultados de precisão da implementação das correções também são muito importantes para confirmar a eficiência do AgentK. O gráfico 2 mostra de forma clara o resultado global de sucesso e erro das implementações, caracterizando como sucesso apenas os cenários em que os erros foram totalmente sanados e o respectivo *pod* passou a rodar normalmente após a correção.

Gráfico 2: Resultado global dos testes de correção automática nos arquivos YAML.



Fonte: elaboração dos autores (2025)

Para cada um dos processos de requisição, é possível obter a quantidade de *tokens* de entrada, a quantidade de *tokens* de saída e o tempo utilizado até receber a resposta. A tabela 3 traz o resultado médio para cada arquivo de exemplo testado; o custo foi multiplicado por dez para se chegar a valores mensuráveis e de melhor entendimento.

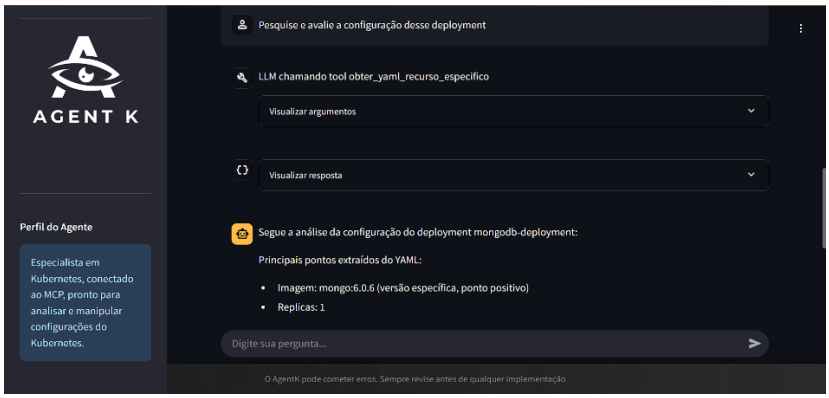
Tabela 3 - Desempenho com a utilização do AgentK em sua nova arquitetura.

Arquivo	Média tokens de entrada	Média tokens de saída	Tempo médio de resposta	Custo / 10 sessões no AgentK (US$)
1-orion.yaml	6998	153	0,29	0,15
2-frontend.yaml	22070	730	0,74	0,50
3-mysql.yaml	9675	1467	3,94	0,31
4-vllm.yaml	11622	663	0,93	0,29
5-nginx.yaml	12133	449	0,7	0,28
6-selenium.yaml	20618	669	1,27	0,47
7-elasticsearch.yaml	6244	58	0,37	0,13
8-newrelic.yaml	6022	62	0,31	0,13
9-storm.yaml	5903	137	0,47	0,13
10-mongodb.yaml	5722	58	0,46	0,12

Fonte: elaboração dos autores (2025)

A figura 5 ilustra o AgentK em operação no ambiente de teste, demonstrando a análise de um *deployment* ativo no *cluster* Kubernetes. Nessa etapa, o sistema identifica boas práticas de configuração, como o uso adequado de *labels*, a definição explícita de *requests* e *limits* de recursos e a consistência das informações no metadado. O AgentK processa o arquivo YAML e apresenta o resultado de forma estruturada e compreensível, evidenciando sua capacidade de interpretação contextual e validação automática de boas práticas.

Figura 5: AgentK em funcionamento.



Fonte: elaboração dos autores (2025)

Considerações finais

O AgentK demonstrou-se como alternativa inovadora de aplicação de LLM ao Kubernetes, oferecendo identificação e correção inteligente de erros em arquivos YAML. Os resultados confirmaram que a integração do MCP com interface conversacional detecta erros de configuração de forma prática e eficiente, permitindo implementação automática de correções. Contudo, o estudo revelou pontos que necessitam de de melhoria: ampliação de cobertura de segurança, experimentação com diferentes LLM e testes envolvendo outros tipos de *misconfigurations*.

O AgentK constitui contribuição relevante para a aplicação de Inteligência Artificial em Kubernetes, consolidando-se como campo fértil para investigações futuras em infraestrutura e cidades inteligentes.

Referências

ALHAMAZANI, Khalid; *et al.* An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art. **arXiv**, 2013. Disponível em: https://arxiv.org/abs/1312.6170. Acesso em: 12 maio 2025.

BBC NEWS. Tesla investigates claims of crypto-currency hack. 2018. Disponível em: https://www.bbc.com/news/technology-43140005. Acesso em: 12 maio 2025.

BEENA, G. M.; RAJENDRAN, N.; THANGARAJ, M.; THOMAS, C. D. Adaptive energy optimization in cloud computing through containerization. **IEEE Access**, v. 13, p. 10478-10491, 2025. DOI: https://doi.org/10.1109/ACCESS.2025.3536231. Disponível em: https://ieeexplore.ieee.org/document/10888964. Acesso em: 13 set. 2025.

BUCHANAN, Randy. Smart cities – a structured literature review. **Smart Cities**, v. 6, n. 4, p. 1719-1743, 2023. DOI: https://doi.org/10.3390/smartcities6040080. Acesso em: 17 set. 2025.

BUSINESS RESEARCH INSIGHTS. Kubernetes Solutions Market. Disponível em: https://www.businessresearchinsights.com/pt/market-reports/kubernetes-solutions-market-101114. Acesso em: 31 maio 2025

CLOUD NATIVE COMPUTING FOUNDATION. Cloud Native 2024: Approaching a Decade of Code, Cloud, and Change – 2024 CNCF Annual Survey. San Francisco: CNCF / The Linux Foundation, abril 2025. Disponível em: https://www.cncf.io/wp-content/uploads/2025/04/cncf_annual_survey24_031225a.pdf.Acesso em: 08 nov. 2025.

GLASPER, Gustav. kube-score: Kubernetes object analysis with recommendations for improved reliability and security. [S.l.: s.n.], 2025. Disponível em: https://github.com/zegl/kube-score. Acesso em: 20 junho 2026.

GRACIAS, Jose Sanchez; PARNELL, Gregory S.; SPECKING, Eric; POHL, Edward A.;

HAMON, Yann. Kubeconform: *a fast Kubernetes manifests validator, with support for Custom Resources*. [S.l.: s.n.], 2025. Disponível em: https://github.com/yannh/kubeconform. Acesso em: 20 junho 2026.

HOU, Zhenyu; LI, Ming; ZHANG, Wei; CHEN, Kai. *Model Context Protocol (MCP): Landscape, security threats, and opportunities*. **arXiv preprint** arXiv:2503.09876, 2025. Disponível em: https://arxiv.org/abs/2503.09876. Acesso em: 14 set. 2025.

JIANG, Y.; SHAO, Y.; MA, D.; SEMNANI, S.; LAM, M. Into the unknown unknowns: engaged human learning through participation in language model agent conversations. **arXiv** preprint arXiv:2408.15232, 2024. Disponível em: https://arxiv.org/abs/2408.15232. Acesso em: 3 nov. 2025.

KUBERNETES. **Kubernetes documentation**. [S.l.: s.n.], [s.d.-a]. Disponível em: https://kubernetes.io/docs/home/. Acesso em: 10 maio 2025.

KUBERNETES. Examples. [S.l.: s.n.], [s.d.-b]. Disponível em: https://github.com/kubernetes/examples. Acesso em: 10 nov. 2025.

KRUGER, Juliano Milton. **Metodologia da pesquisa em Administração**: em linguagem descomplicada. 1. ed. Curitiba: Editora Bagai, 2023. ISBN 978-65-5368-212-2.

LUKŠA, Marko. **Kubernetes in action**. Shelter Island: Manning Publications, 2018. ISBN 978-1-61729-372-6.

MATTHIAS, Karl; KANE, Sean P. **Primeiros passos com Docker**: usando contêineres em produção. Tradução de Alison Miazaki. 1. ed. São Paulo: Novatec Editora, 2016. ISBN 978-85-7522-473-1.

MINAEE, Shervin; et al. Large language models: a survey. **arXiv**, 2025. Disponível em: https://arxiv.org/abs/2402.06196. Acesso em: 8 maio 2025.

MINIKUBE. Documentação do Minikube. [S.l.: s.n.], 2025. Disponível em: https://minikube.sigs.k8s.io/docs/. Acesso em: 8 maio 2025.

MURALIDHARAN, Shapna; SONG, Gyuwon; KO, Heedong. Monitoring and managing IoT applications in smart cities using Kubernetes. In: INTERNATIONAL CONFERENCE ON SMART CITIES, CLOUD, BIG DATA EMERGING TOPICS – SMART 2019, 2019, Venice. Anais [...]. [S.l.]: IARIA, 2019. p. 1–6. ISBN 978-1-61208-703-0. ISSN 2308-4294. Disponível em: https://www.thinkmind.org/index.php?view=article&articleid=smart_2019_1_20_80010. Acesso em: 11 maio 2025.
NIEDERMAIER, Manuel; et al. On observability and monitoring of distributed systems: an industry interview study. **arXiv**, 2019. Disponível em: https://arxiv.org/abs/1907.12240. Acesso em: 16 set. 2025.
NIST. The NIST definition of cloud computing. Gaithersburg: National Institute of Standards and Technology, 2011. (Special Publication 800-145).
OLIMPIO, Vinicius Gabriel da Silva; SANTANA, Cleber dos Santos; NEVES, David Santos Pinheiro das; SILVA, Wilson Pereira da. AgentK. São Bernardo do Campo: [s.n.], 2025a. Disponível em: https://github.com/viniolimpio3/AgentK/. Acesso em: 10 nov. 2025.
OLIMPIO, Vinicius Gabriel da Silva; SANTANA, Cleber dos Santos; NEVES, David Santos Pinheiro das; SILVA, Wilson Pereira da. AgentK-MCP. São Bernardo do Campo: [s.n.], 2025b. Disponível em: https://github.com/viniolimpio3/AgentK-MCP/. Acesso em: 10 nov. 2025.
OPENAI. OpenAI Platform. San Francisco: OpenAI, 2025. Disponível em: https://platform.openai.com. Acesso em: 17 nov. 2025.
OPEN POLICY AGENT. Conftest: write tests against structured configuration data using the Open Policy Agent Rego query language. [S.l.: s.n.], 2025. Disponível em: https://github.com/open-policy-agent/conftest. Acesso em: 20 de junho de 2026.
POURMAJIDI, Amir; et al. On challenges of cloud monitoring. **arXiv**, 2018. Disponível em: https://arxiv.org/abs/1806.05914. Acesso em: 16 set. 2025.
RAY, João. A survey on model context protocol. **arXiv** preprint arXiv:2501.12345, 2025. Disponível em: https://arxiv.org/abs/2501.12345. Acesso em: 15 set. 2025.
SENEL, Berat Can; MOUCHET, Maxime; CAPPOS, Justin; FOURMAUX, Olivier; FRIEDMAN, Timur; MCGEER, Rick. Multitenant Containers as a Service (CaaS) for Clouds and Edge Clouds. **arXiv**, 2023. Disponível em: https://arxiv.org/pdf/2304.08927.pdf. Acesso em: 17 set. 2025.
STATE of Kubernetes security report 2022. [S.l.: s.n.], 2022. Disponível em: https://www.redhat.com/rhdc/managed-files/cl-state-of-kubernetes-security-report-2022-ebook-f31209-202205-en.pdf. Acesso em: 20 fev. 2025.
UN-HABITAT. **World Cities Report 2022**: Envisaging the Future of Cities. Nairobi United Nations Human Settlements Programme, 2022. Disponível em: https://unhabitat.org/world-cities-report-2022. Acesso em: 17 set. 2025.
UNITED NATIONS. The World's Cities in 2018 – Data Booklet. New York: United Nations, Department of Economic and Social Affairs, Population Division, 2018. (ST/ESA/SER.A/417). Disponível em: https://www.un.org/en/events/citiesday/assets/pdf/the_worlds_cities_in_2018_data_booklet.pdf . Acesso em: 12 maio 2025.
WANG, L.; MA, C.; FENG, X.; ZHANG, Z.; YANG, H.; ZHANG, J.; CHEN, Z.; TANG, J.; CHEN, X.; LIN, Y.; ZHAO, W. X.; WEI, Z.; WEN, J. R. A survey on large language model based autonomous agents. **Frontiers of Computer Science**, v. 18, n. 6, p. 186345, 2024. DOI: https://doi.org/10.1007/s11704-024-40231-1. Disponível em: https://link.springer.com/article/10.1007/s11704-024-40231-1. Acesso em: 3 nov. 2025.
XU, Hanxiang; et al. Large Language Models for Cyber Security: a Systematic Literature Review. arXiv preprint **arXiv**:2405.04760, 2024. Disponível em: https://arxiv.org/abs/2405.04760. Acesso em: 15 set. 2025.
YAML. YAML — YAML Ain't Markup Language™: revision 1.2.2. [S.l.]: YAML.org, 2021. Disponível em: https://yaml.org/. Acesso em: 10 maio 2025.